MPIfR (Max-Planck Institut für Radioastronomie Bonn) &
AIRUB (Astronomisches Institut der Ruhr-Universität Bochum)

Sven A.H. Müller, supported by Marcus Albrecht, Frank Bertoldi, & Frederic Schuller
AIRUB, Bochum, Germany
28. 02. 2005

BoA – The Bolometer Data Analysis Project

0.912

Reference and user's manual

Abstract

BoA is a new software and will be used for the reduction of data obtained with the bolometer arrays at the Atacama Pathfinder Experiment (**APEX**) telescope, at the IRAM 30m telescope and at the Heinrich-Herz Telescope (HHT). **BoA** shall include most of the functionalities of the current reduction packages MOPSI and NIC, in a programming environment which is easier to modify, maintain, and re-use. Unlike the current packages, **BoA** shall naturally interface with an observation management and calibration database, and it shall aid well in hardware commissioning and debugging.
BoA is a collaborative effort of the Astronomisches Institut der Ruhr-Universität Bochum (AIRUB) and the Max-Planck-Institut für Radioastronomie (MPIfR) Bonn, and invites participation by all interested parties.

The initiative to build a new bolometer software package is motivated and financially supported by a "Verbundforschung" grant by the German Ministry of Research and Education (BMBF) to the AIRUB and MPIfR to build LABOCA, a large bolometer array camera for **APEX** .
APEX is a 12-meter radio telescope at the best accessible site for submillimeter observations, Llano de Chajnantor in Chile's Atacama desert. **APEX** is an international collaboration led by the Millimeter and Submillimeter astronomy group at the MPIfR. AIRUB, the European Southern Observatory (ESO) and the Onsala Space Observatory (OSO) agreed with the MPIfR to share **APEX** funding and observing time, with 10% of the observing time set aside for Chilean astronomers. The antenna has been constructed by VERTEX Antennentechnik and erection at the site was finished in spring time of 2004.
This document describes how to reduce LABOCA data and includes detailed descriptions of each task.

Copyright © 2003 – 2005 AIRUB and MPIfR

Contents

1 Introduction **1**

2 Philosophy and basic structure **2**
2.1 Philosophy . . . 2
2.2 Programming language: Python . . . 2
2.3 Basic structure . . . 2

3 Installation **5**
3.1 Preparation . . . 5
3.2 Installation using the *install.sh* script . . . 5
3.2.1 Runnig the *install.sh* script . . . 5
3.3 Installation using the boa_build script . . . 10
3.3.1 Download the boa_build script . . . 10
3.3.2 Runnig the boa_build script . . . 11
3.3.3 Known problems: . . . 15
3.4 Manual installation of each package as user . . . 16
3.4.1 Python 2.3.2 . . . 16
3.4.2 Numeric: . . . 17
3.4.3 numarray . . . 17
3.4.4 pgplot . . . 18
3.4.5 ppgplot . . . 19
3.4.6 Installation of the Intel Fortran Compiler . . . 21
3.4.7 swig . . . 22
3.4.8 pySLALIB . . . 23
3.4.9 cfitsIO . . . 23
3.4.10 Scipy Distutils . . . 24
3.4.11 pcfitsIO . . . 24
3.4.12 Path structure . . . 24
3.4.13 Listing of python site-packages . . . 25
3.4.14 Readline interactive tool . . . 25
3.4.15 Additional Libraries . . . 26
3.5 Updating BoA . . . 26

4 Data organisation **28**
4.1 Input data . . . 28
4.2 Data objects . . . 28
4.2.1 Raw data . . . 28
4.2.2 Derived data . . . 29

4.2.3 Processed data . . . 30
4.2.4 BoA Status . . . 30
4.3 Output data . . . 31

5 Usage **32**
5.1 Starting BoA . . . 32
5.2 Description of how Boa works . . . 32
5.2.1 Methods . . . 32
5.2.2 Arguments . . . 33
5.2.3 Output . . . 34
5.3 Detailed description of all user methods . . . 35
5.3.1 All at once: the *reduce* method . . . 35
5.3.2 Pointing . . . 36
5.3.3 Focus . . . 36
5.3.4 Skydip . . . 36
5.3.5 OnOff . . . 36
5.3.6 Mapping . . . 36
5.3.7 Beam maps . . . 37
5.3.8 Reading a FITS file . . . 37
5.3.9 Opening a plot window . . . 37
5.3.10 Clearing a plot window . . . 38
5.3.11 Closing a plot window . . . 38
5.3.12 Channel Maps . . . 38
5.3.13 azimuth . . . 38
5.3.14 elevation . . . 39
5.3.15 azel . . . 40
5.3.16 channels . . . 41
5.3.17 signal . . . 41
5.3.18 Plotting FFT . . . 42
5.4 MB-Fits to FITS file conversion . . . 42
5.5 Scripts . . . 42
5.6 Abbreviations . . . 43

6 Graphics: BoGLi **45**
6.1 Introduction . . . 45
6.2 Commands . . . 45
6.3 Commands . . . 47
6.3.1 Opening a plot window . . . 47
6.3.2 Closing a plot window . . . 49
6.3.3 Clearing a plot window . . . 49

6.3.4 Selecting a device . . . 49
6.3.5 Resizing a device . . . 50
6.4 Plotting: single plots . . . 50
6.5 Plotting: plot multiple channels . . . 52
6.6 Drawing on an image . . . 53
6.7 Drawing on plots of multiple channels . . . 56
6.8 Device handling . . . 57
6.9 Attributes . . . 58
6.9.1 Basic concepts . . . 58
6.9.2 Float and integer parameters . . . 58
6.9.3 General attribute keywords . . . 59
6.9.4 Affected part keywords . . . 60
6.9.5 Parameter list . . . 61
6.9.5.1 Logical parameters . . . 63
6.9.5.2 String parameters . . . 63
6.9.5.3 List parameters . . . 63

7 Development **64**
7.1 Basic programming rules . . . 64
7.2 Adding classes . . . 64
7.3 Adding methods . . . 64
7.4 Adding Fortran90 code . . . 64
7.5 Interfacing . . . 68
7.6 ScientificPython-2.4.5 . . . 68
7.6.1 Scientific.Statistics.median . . . 70
7.6.2 Scientific.Statistics.mean . . . 71
7.6.3 Scientific.Statistics.correlation . . . 72
7.6.4 Scientific.Functions.LeastSquares . . . 72

8 Description of all classes and methods **74**
8.1 chanana . . . 75
8.1.1 flag_pc(xflag,xdata,n_phase,n_channel,low,high,ii,jj,kk) . . . 76
8.1.2 flag(xflag,xdata,low,high,mask_count,ii) . . . 77
8.1.3 unflag(xflag,xdata,flag,mask_count,ii) . . . 77
8.1.4 flagtime(xflag,low,high,flag,mask_count,ii) . . . 77
8.1.5 flaglon(xflag,low,high,flag,mask_count,ii) . . . 78
8.2 BoaConfig (module) . . . 78
8.2.1 setInDir(inputDirectory=",r=0): . . . 78
8.2.2 setOutDir(outputDirectory=",r=0): . . . 79
8.2.3 listInDir(): . . . 79

8.2.4 setInFile(inputFile='?',r=0): . . . 79
8.2.5 setOutFile(outputFile='?',r=0): . . . 79
8.3 data . . . 80
8.4 DataAna(BoaDataEntity.DataEntity): . . . 82
8.4.1 __init__(self): . . . 82
8.4.2 flagChanFromList(self,list=[],flag=1): . . . 82
8.4.3 unflag(self,channel=-1,phase=-1,flag=1): . . . 83
8.4.4 flag(self,channel=-1,phase=-1,below=2,above=2): . . . 83
8.4.5 flagLST(self,channel=-1,phase=-1,below='?',above='?',flag=1): . . . 83
8.4.6 flagLon(self,channel=-1,phase=-1,below='?',above='?',flag=1): . . . 84
8.4.7 statistics(self): . . . 84
8.4.8 corMatrix(self): . . . 84
8.4.9 correlate(self,phase=-1,channel=1,plot=1): . . . 85
8.4.10 snf(self,phase=-1,channel=1,subtract=1,wa=0.95,wb=1.0): . . . 85
8.4.11 basePolySubscan(self,channel=-1,phase=-1,order=1,subtract=1,plot=0): . . . 85
8.4.12 basePoly(self,channel=-1,phase=-1,order=1,subtract=1,plot=0): . . . 86
8.4.13 addPoly(self,channel=0,phase=1,poly=[10.,0.,10.,10.]): . . . 86
8.5 DataEntity: . . . 87
8.5.1 __init__(self): . . . 87
8.5.2 reset(self): . . . 87
8.5.3 __str__(self): . . . 88
8.5.4 read(self,inFile='',readObs=[],update=0): . . . 88
8.5.5 fillFromMBFits(self,obsEntity,update=0): . . . 88
8.5.6 FillF90(self): . . . 89
8.5.7 dumpData(self,fileName='BoaData.sav'): . . . 89
8.5.8 restoreData(self,fileName='BoaData.sav'): . . . 89
8.5.9 saveMambo(self,inName='',outName=''): . . . 89
8.5.10 readRCPfile(self,rcpFile): . . . 90
8.5.11 writeRCPfile(self,rcpFile='rcpBoa.rcp'): . . . 90
8.5.12 initPhases(self): . . . 90
8.5.13 phaseDiff(self): . . . 91
8.5.14 copyPhases(self): . . . 91
8.5.15 existData(self): . . . 91
8.5.16 getTotalChanList(self): . . . 91
8.5.17 setCurrChanList(self,chanList='?'): . . . 92
8.5.18 printCurrChanList(self): . . . 92
8.5.19 checkChanList(self,inList): . . . 92
8.5.20 getChanData(self,dataType='',chan=1,phase=0,flag=0): . . . 92
8.5.21 getChanListData(self,type='',chanList=[],phase=0,flag=0): . . . 93
8.5.22 signal(self,chanList=[],phase=0,flag=0,\ . . . 93

8.5.23 plotCorrel(self,chanRef=1,chanList=[],phase=0,flag=0,\ . . . 93
8.5.24 azimuth(self,chanList=[],phase=0,flag=0,\ . . . 94
8.5.25 elevation(self,chanList=[],phase=0,flag=0,\ . . . 94
8.5.26 azel(self,chanList=[],phase=0,flag=0,\ . . . 94
8.5.27 eloff(self,chanList=[],phase=0,flag=0,\ . . . 95
8.5.28 azoff(self,chanList=[],phase=0,flag=0,\ . . . 95
8.5.29 azeloff(self,chanList=[],phase=0,flag=0,\ . . . 95
8.5.30 plotMean(self,chanList=[],phase=0,flag=0,\ . . . 95
8.5.31 plotRms(self,chanList=[],phase=0,flag=0,\ . . . 96
8.5.32 plotMeanChan(self,chanList=[],phase=0,flag=0,\ . . . 96
8.5.33 plotRmsChan(self,chanList=[],phase=0,flag=0,\ . . . 96
8.5.34 chanMap(self,chanList=[],phase=0,flag=0,oversamp=2.): . . . 97
8.5.35 slowChanMap(self,chanList=[],phase=0,flag=0,oversamp=2.,sizeX=[],sizeY=[]): 97
8.5.36 fastChanMap(self,chanList=[],phase=0,flag=0,oversamp=2.): . . . 97
8.5.37 fft(self,chanList=[],phase=0,flag=0,\ . . . 97
8.6 f1 . . . 98
8.6.1 gsmooth(image,fwhm,ii,jj) . . . 99
8.6.2 reimage(image,data,xy,dxy,zero_offset,xy_scale,ii,jj,kk) . . . 100
8.6.3 compress(xdata,xflag,flag_value,ii) . . . 101
8.6.4 get_subscan_index(xsubscan_index,ii) . . . 101
8.7 Focus: . . . 103
8.7.1 __init__(self,BoaB): . . . 103
8.7.2 scanFocus(self,chan=-1,phase=0,flag=0,plot=1,omit=[],r=0): . . . 104
8.7.3 accFocus(self,com='?',r=0): . . . 104
8.7.4 detSubStruct(self,chan=1,flag=0): . . . 105
8.7.5 detScanExtr(self): . . . 105
8.7.6 createModelData(self): . . . 105
8.7.7 printScanResults(self): . . . 106
8.7.8 initAccFocus(self): . . . 106
8.7.9 addToAccFocus(self): . . . 106
8.7.10 detAccExtr(self): . . . 107
8.7.11 createAccModelData(self): . . . 107
8.7.12 printAccResults(self): . . . 107
8.8 baseline . . . 108
8.9 Image: . . . 109
8.9.1 wcs2pix(self,X,Y): . . . 109
8.9.2 wcs2phy(self,i,j): . . . 109
8.9.3 computeWCS(self,pixelSize,sizeX=[],sizeY=[],minmax=[]): . . . 109
8.9.4 physicalCoordinates(self): . . . 110
8.9.5 display(self,rms=0,weight=0,\ . . . 110

8.10 Map(BoaDataAnalyser.DataAna): . . . 111
8.10.1 __init__(self): . . . 111
8.10.2 slowMap(self,chanList=[],phase=0,flag=0,oversamp=2.0,beammap=0,\ . . . 111
8.10.3 slowMap2(self,chanList=[],phase=0,flag=0,oversamp=2.0,beammap=0,\ . . . 111
8.10.4 fastMap(self,chanList=[],phase=0,flag=0,oversamp=2.0,beammap=0,\ . . . 112
8.10.5 showMap(self,style='g2r',labelX="\ gDAz["]",labelY="\ gDEl["]",wedge=1,limitsZ=[]): . . . 112
8.10.6 beamMap(self,chanList=[],phase=0,flag=0,oversamp=2.0,sizeX=[],sizeY=[]): . . . 113
8.10.7 fastBeam(self,chanList=[],phase=0,flag=0,oversamp=2.0,\ . . . 113
8.10.8 wcs2pix(self,X,Y): . . . 113
8.10.9 wcs2phy(self,i,j): . . . 114
8.10.10 computeWCS(self,pixelSize,sizeX=[],sizeY=[],minmax=[]): . . . 114
8.10.11 boloPos(self,chanList,allFlux): . . . 114
8.10.12 getPixel(self,nbPix=3): . . . 114
8.11 MessHand: . . . 115
8.11.1 __init__(self,logName='Unknown'): . . . 115
8.11.2 Welcome(self): . . . 115
8.11.3 setMaxWeight(self,weight='2'): . . . 116
8.11.4 error(self,message="): . . . 116
8.11.5 warning(self,message="): . . . 116
8.11.6 info(self,message="): . . . 116
8.11.7 longinfo(self,message="): . . . 117
8.11.8 debug(self,message="): . . . 117
8.11.9 setMess(self,weight=1,message="): . . . 117
8.11.10 initMessFile(self,filename="boa.mes"): . . . 117
8.11.11 closeMessFile(self): . . . 118
8.12 OnOff: . . . 119
8.12.1 __init__(self,BoaB): . . . 119
8.13 Point(BoaMapping.Map): . . . 120
8.13.1 __init__(self): . . . 122
8.13.2 iterMap(self,chanList=[],phase=0,flag=0,sizeX=[],sizeY=[]): . . . 123
8.13.3 solvePointing(self,chanList=[],gradient=1,radius=0): . . . 123
8.13.4 showPointing(self,plot=1): . . . 123
8.14 snf . . . 124
8.15 Sky: . . . 125
8.15.1 __init__(self,BoaB): . . . 125
8.16 test1 . . . 126
8.16.1 flag(n_ch,n_ph,low,high) . . . 126
8.17 MamboMBFits: . . . 127
8.17.1 __init__(self,mamboName,mbName): . . . 128

8.17.2 convertMambo2MBFits(self): . . . 128
8.17.3 convertMB2MamboFits(self): . . . 128
8.17.4 readMambo(self): . . . 129
8.17.5 readMBfits(self): . . . 129
8.17.6 initMambo(self): . . . 129
8.17.7 initMB(self): . . . 129
8.17.8 convertMamboPrimary(self): . . . 130
8.17.9 fillMamboPrimary(self): . . . 130
8.17.10 fillFebepar(self): . . . 130
8.17.11 convertFebepar(self): . . . 130
8.17.12 processMamboData(self): . . . 131
8.17.13 fillMamboData(self): . . . 131
8.17.14 getScanType(self): . . . 131
8.17.15 getObsMode(self): . . . 131
8.17.16 readRCP(self): . . . 132
8.17.17 sbas2ctype(self): . . . 132
8.17.18 ctype2sbas(self): . . . 132
8.18 Timing: . . . 133
8.18.1 safeExp(x): . . . 133
8.18.2 fitParabola(x,y,err): . . . 133
8.18.3 detStartParaParabola(x,y): . . . 134
8.18.4 parabola(p,fjac=None,x=None,y=None,err=None): . . . 134
8.18.5 modelparabola(p,x): . . . 134
8.18.6 draw2Dgauss(p): . . . 134
8.18.7 modelBase2Dgauss(p,position): . . . 135
8.18.8 base2DGauss(p,fjac=None,x=None,y=None,err=None): . . . 135
8.18.9 fitBase2DGauss(mapArray,x,y,err=1.0,fwhm=11.0,gradient=1): . . . 135
8.18.10 cropped_circular_gaussian(p,position,threshold=3): . . . 136
8.18.11 gaussian(r2,sig2): . . . 136
8.18.12 distsq(x1,y1,x2,y2): . . . 136
8.18.13 solvePoly(order,dataX,dataY): . . . 136
8.18.14 module2(c): . . . 137
8.18.15 mod_fft(z,optimize=0): . . . 137
8.19 fUtilities (Fortran module) . . . 137
8.19.1 ksmooth(image,ii,jj,kernel,ngg) . . . 137
8.20 BoGLi modules and classes . . . 138
List of Figures139 List of Tables140

1 Introduction

The Atacama Pathfinder Experiment (APEX)[1] is a 12-meter radio telescope at the best accessible site for submillimeter observations, Llano de Chajnantor in Chile's Atacama desert.

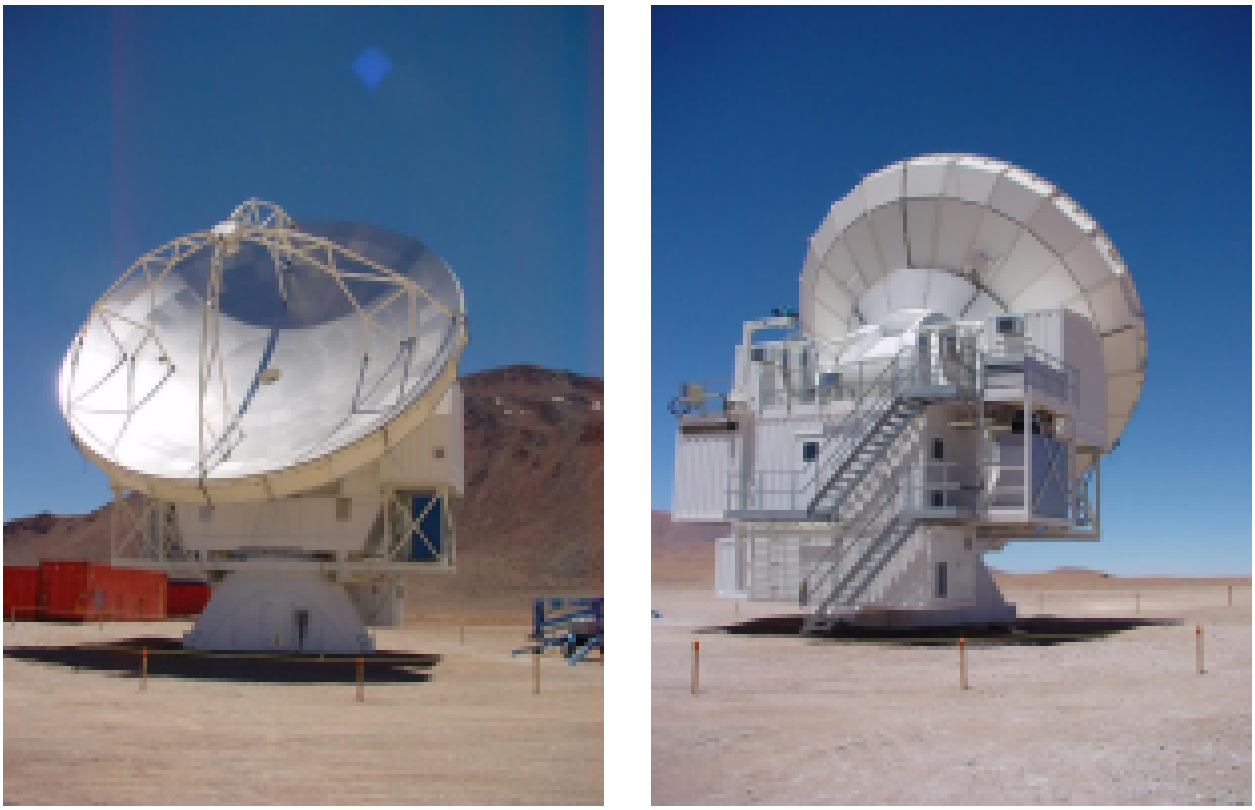

Figure 1.1: The APEX telescope at Chajnantor, Images from the APEX handover, Nov. 2003

[1] http://www.mpifr-bonn.mpg.de/div/mm/apex/

2 Philosophy and basic structure

2.1 Philosophy

BoA is designed with essentially two goals in mind: a) to provide a most comprehensive tool for the reduction and analysis of data from the new generation of bolometer arrays and b) to facilitate the extension and modification of the software by any user. BoA is intended to combine a simple and intuitive usage with the coverage of all aspects of data reduction from raw data to final images. It shall provide the possibility to insert new functionalities as well as to be included in other software contexts. This also results in a facile maintenance by others than the developers.
The natural choice for the creation of BoA is object oriented programming. A simplistic approach of object orientation would be to glue everything together: data and methods. Nevertheless, the large variety of required functionalities would result in data objects containing a huge amount of methods, either by definition within the data class or by inheritance. This would lead to a rather complex structure and inhibit easy extension of the software. Moreover, some functionalities need to work on a whole set of data while others do not have any relation to these at all. The chosen approach is therefore to use a data class containing some basic methods for filling the resulting objects while the remaining functionalities are merged in a set of classes with a straightforward subdivision of responsibilities.

2.2 Programming language: Python

As programming language Python is used. Python is an interpreted, interactive and object oriented language though it does not adhere to all concepts of object orientation as strictly as e.g. C++ does. The resulting shortcomings have to be kept in mind when extending BoA and can be overcome by sticking to some basic programming rules. Python is a scripting language and as such allows a fast and easy extension of BoA by the user. It facilitates the wrapping of code written in C/C++ or Fortran90.

2.3 Basic structure

BoA consists of a set of classes, most of which are defined in dedicated modules (files). In addition, a few functions are defined in separate modules. A detailed description of all classes and methods can be found in Sect. 8. The subdivision was chosen to reach a high modularity and an obvious compilation of assortative functionalities within one class. Two kinds of classes may be distinguished:

- Data classes: The DataEntity class defines the data structure which is used within BoA. Objects of this class contain the raw and reduced data and all relevant parameters of a single scan. This class also defines methods to fill the data object from an MBFITS file. Then, the DataAna class inherits from DataEntity: it contains all data related methods, plus some methods for data analysis (e.g. flagging, baseline). Then, the Map class inherits from DataAna: it contains all methods defines in DataEntity and DataAna, plus specific methods for map processing and

display. Finally, classes dedicated to various observing modes inherit from the Map class: they contain additional methods specific to a given type of observation. Table 1 lists BoA data classes, with module names and short descriptions of their responsibilities.

- Peripheral classes: All other classes provide methods which either are used by data objects (e.g. Image is used within Map objects), or provide functionalities on the BoA level (e.g. MessHand). These classes are summarized in Table 2.

Table 1: BoA data classes

class name	module	purpose
DataEntity	BoaDataEntity.py	data and parameters storage
DataAna	BoaDataAnalyser.py	general data analysis methods
Map	BoaMapping.py	map reduction
Focus	BoaFocus.py	focus reduction
OnOff	BoaOnOff.py	onoff reduction
Point	BoaPointing.py	pointing reduction
Sky	BoaSkydip.py	skydip reduction

Table 2: Other BoA classes

class name	module	purpose
Image	BoaMapping.py	image and axis description
CalData	BoaCalibrationData.py	calibration data
CorrCal	BoaCorrCal.py	correction and calibration
FitsOut	BoaFitsOutput.py	FITS output
Help	BoaHelp.py	online help
MessHand	BoaMessageHandler.py	message handling
DatabaseConn	BoaDatabaseConnector.py	connection to database
MamboMBFits	MamboMBFits.py	MAMBO to/from MB-Fits conversion
Timing	Utilities.py	benchmarking utilites

Finally, a few functions are defined in separate modules (listed in Table 3), which do not define any class. Thus, these functions can easily be imported and run from any level. In particular, the BoA Graphic Library (BoGLi) is defined in a collection of modules, which can be imported at the python level and do not require BoA. A description of BoGLi is given in Sect. 6.
In addition, a number of utility and computing routines are written in Fortran modules. These routines are used within Python methods, and should in principle not be called directly by a BoA user.

Table 3: Other BoA modules

module name	purpose
BoGLi (see Sect. 6)	Graphic library
Utilities.py (see Sect. 8.18)	collection of utilities
BoaConfig.py (see Sect. 8.2)	global parameters definitions
BoaSimulation.py	LABOCA data simulator

3 Installation

[CV: General comment:- This document should say somewhere from where the reader should get/have got boa from, i.e. download etc. CVS should also be mentioned in/prior to this section. Unless I've missed these points, I don't think are mentioned prior to section 3 and so is "assumed" reader is already familiar with what they are supposed to do and where it comes from, even what "cvs" is.]

This section describes how to install BoA and all required additional software packages.
In the first subsection about preparation all additional needed software packages are named together with a short description of their installation procedures. The second part describes the installation using the installation script of Alexandre Beelen and Frederic Schuller, called *install.sh*, which is the prefered method. The third part describes the installation using the older, now obsolete installation script of Frederic Schuller, called *boa_build*. The fourth part deals with the manual installation of all software as user. Recommended is the installation via the *install.sh* script. In the last section you will learn how to update an existing BoA version.

3.1 Preparation

Firstly, you will need to install a GNU C compiler (e.g. gcc and g77), the Xfree development kit and the libpng library, which all are normally included in every linux distribution.

3.2 Installation using the *install.sh* script

This subsection describes how to use the installation script *install.sh* written by Alexandre Beelen, Thomas Jürges and Frederic Schuller. You can find it in the directory /install of the openboa cvs directory.
[CV: This section doesn't seem clear to me, especially last line...install.sh is in openboa/install, true, but I find this confusing:- should the reader at this point know what "openboa cvs" means? Also need a prior mention of cvs (see general comment, above).]

3.2.1 Runnig the *install.sh* script

Before runnig the *install.sh* installation script you have to make sure that you fulfill following prerequisites:
The Gnu C-compiler *gcc* (e.g. gcc and g77), the Xfree development kit and the libpng library must be available. They normally come with every Linux distribution.
Now go to the directory where you have downloaded the openboa cvs directory and files. Change into the directory openboa/install/ where the installation script *install.sh* is stored and run it:

```
./install.sh
```

You will be prompted to enter yes (y) or no (n) for the installation of each single software package. Attention, at some points of the installation you don't need to confirm the input of yes (y) or no (n) with the return key, so just press the y-key and see the installation running on (i.e. for the installation of the Intel Fortran 90 compiler, for F2PY, for).
If you don't want to use the default installation path, then please enter **your chosen** path, e.g. */home/smueller/BoA*, **when prompted. Don't forget to create your chosen directory if not already present. [CV: See changes above.]**
The script will create in this installation directory six sub-directories, *bin*, *BoA*, *include*, *lib*, *man* and *tmp* where all necessary files will be installed. The required disk space is about 220 MB.
[CV: As far as I know, in my case, boa sub-dir did not end up in same place as the rest listed above...is that intentional? (Also see Section 3.3.2).]
After the installation please execute the following files:

```
source .boarc.sh       (located in your home directory)
```

and

```
source start_boa.sh    (located in your BOA directory)
```

You can now run *BoA* by entering **boa** at the linux prompt, e.g.:

```
cd /home/smueller/BOA/fits/
boa
```

Example: Installation of the *BoA-Software* into the directory /home/observer/BOA/

```
observer@thora:~> cd openboa/install/
observer@thora:~/openboa/install> ./install.sh
This program will install the Bolometer Analysis Package
You can find a logfile of the installation in
/home/observer/openboa/install/build.stat

* In which directory do you want to install the BoA softwares?
(/home/observer) : /home/observer/BOA
* In which directory are the external packages located?
(/home/observer/openboa/install/ExtPkg/) :

Starting installation in /home/observer/BOA

Do you want to install Python-2.3.2 (version 2.2.2 required) (y/N)? y
```

```
<boa_build> Unpacking and installing Python-2.3.2
<boa_build>  This may take some time. In case of crash check files in
<boa_build>  /home/observer/BOA/tmp
<boa_build>   -----  Python-2.3.2 : Done -----
Do you want to install Swig-1.3.23 (y/N)? y
<boa_build> Unpacking and installing Swig-1.3.23
<boa_build>  This may take some time. In case of crash check files in
<boa_build>  /home/observer/BOA/tmp
<boa_build>   ----- swig-1.3.23 : Done -----
Do you want to install Numeric-23.1 (version 23.1 required) (y/N)? y
<boa_build> Unpacking and installing Numeric-23.1
<boa_build>  This may take some time. In case of crash check files in
<boa_build>  /home/observer/BOA/tmp
<boa_build>   -----  Numeric-23.1 : Done -----
Do you want to install numarray-0.9 (version 0.8 required) (y/N)? y
<boa_build> Unpacking and installing numarray-0.9

<boa_build>   -----  numarray-0.9 : Done -----
Do you want to install pgplot5.2 (version 5.2 required) (y/N)? y
<boa_build> Unpacking and installing pgplot-5.2
For additional information, read file ./sys_linux/aaaread.me
Reading configuration file: ./sys_linux/g77_gcc.conf
Selecting uncommented drivers from ./drivers.list
Found drivers GIDRIV HGDRIV LSDRIV LXDRIV NUDRIV PGDRIV PSDRIV TTDRIV X2DRIV XWDRIV
Creating make file: makefile
Determining object file dependencies.
<boa_build>   -----  re-running makemake with included drivers.list -----
For additional information, read file ./sys_linux/aaaread.me
Reading configuration file: ./sys_linux/g77_gcc.conf
Selecting uncommented drivers from ./drivers.list
Found drivers GIDRIV HGDRIV LSDRIV LXDRIV NUDRIV PGDRIV PSDRIV TTDRIV X2DRIV XWDRIV
Creating make file: makefile
Determining object file dependencies.
gcc -Wall -fPIC -DPG_PPU -O -I. ./cpg/pgbind.c -o pgbind
./pgbind bsd -h -w ./src/pgarro.f ./src/pgask.f  ./src/pgaxis.f ./src/pgaxlg.f ./src/pgband.f
gcc -c -Wall -fPIC -DPG_PPU -O -I. cpg*.c
rm -f cpg*.c
ar ru libcpgplot.a cpg*.o
ar: creating libcpgplot.a
ranlib libcpgplot.a
```

```
rm -f cpg*.o
gcc -Wall -O -c -I. ./cpg/cpgdemo.c
g77 -o cpgdemo cpgdemo.o -L‘pwd‘ -lcpgplot -lpgplot -L/usr/X11R6/lib -lX11
rm -f cpgdemo.o

*** Finished compilation of the C PGPLOT wrapper library ***

Note that if you plan to install the library in a different
directory than the current one, both libcpgplot.a and cpgplot.h
will be needed.

<boa_build>   -----  pgplot-5.2 : Done -----
Do you want to install pPGPLOT1.3 (version 1.3 required) (y/N)? y
<boa_build> Unpacking and installing pPGPLOT
<boa_build>   -----  pPGPLOT-1.3 : Done -----
Do you want to install slalib  (y/N)? y
<boa_build> Unpacking and installing slalib
<boa_build>   -----  slalib : Done -----
Do you want to install pySLALIB-0.4 (version 0.4 required) (y/N)? y
<boa_build> Unpacking and installing pySLALIB-0.4
<boa_build>   -----  pySLALIB-0.4 : Done -----
Do you want to install cFITSIO-2.49 (version 2.44 required) (y/N)? y
<boa_build> Unpacking and installing cFITSIO...
<boa_build>   -----  cFITSIO-2.49 : Done -----
Do you want to install p_CFITSIO (very specific version!) (y/N)? y
<boa_build> Installing the included version of pCFITSIO
<boa_build>   -----  p_CFITSIO : Done -----
Do you want to install the Intel Fortran 90 compiler (y/n)? y
<boa_build> Unpacking and installing Intel F90
~/BOA/tmp ~/openboa/install
~/BOA/tmp/intel_fc_80 ~/BOA/tmp ~/openboa/install
~/BOA/tmp ~/openboa/install
~/BOA/tmp/intel_idb_80 ~/BOA/tmp ~/openboa/install
~/BOA/tmp ~/openboa/install
~/openboa/install
<boa_build>   -----  Intel F90 : Done -----
Mon Feb 28 14:03:00 CET 2005
Do you want to install F2PY (cvs latest... you better install this one) (y/N)? y
<boa_build> Unpacking and installing F2PY
<boa_build> You need to be connected if you want to retrieve the latest version of scipy_distu
```

```
<boa_build> If asked for a password use : 'guest'
Do you want to checkout from CVS (y/N)? N
<boa_build>   -----  F2PY-2-latest : Done -----
Do you want to install python mpfit (y/N)? y
Do you want to install the 'interactive' python script (y/N)? y
Do you want to install the latest version of BoA (y/N)? y
Where would you like to install boa
(/home/observer/BOA/boa) :
Shall CVS be used for BoA sources (y/N) ?y
Enter you login to the cvs server: mueller
Password:
<boa_build> Building the Fortran module (ifc needed)
Do you want to install example MB-Fits files (y/N)? y
Where would you like to install ezample MB-Fits files
(/home/observer/BOA/fits) :
Password:
Do you want to install the documentation (y/N)? y
Where would you like to install the documentation
(/home/observer/BOA/doc) :
Password:
<boa_build> Copying .boarc.[c]sh.new shell files to ~
~/openboa/install ~/BOA/tmp
<boa_build>   -----  .boarc : Done -----
Mon Feb 28 14:10:23 CET 2005
Do you want to create a startup file (y/N)? y
Where would you like to install the startup file
(/home/observer/BOA/start_boa.sh) :
<boa_build>   -----  Latest BoA : Done -----
<boa_build> Installation completed at:Mon Feb 28 14:12:36 CET 2005
<boa_build> You may want to remove/check files in directory:
<boa_build> /home/observer/BOA/tmp/
```

In case of the on-line installation you will need for the installation of F2PY to enter the password "guest":

```
Getting and installing F2PY
Press Enter for password
Logging in to :pserver:anonymous@scipy.org:2401/home/cvsroot
CVS password: guest
```

If one or more of the servers providing the software sources is down or cannot be reached you can alternativly download all needed software packages from the LABOCA web page at the AIRUB. Just copy all archives into the directory *boa_build* and run the script *./boa_build* again.

3.3 Installation using the boa_build script

This subsection describes how to use the installation script *boa_build* written by Frederic Schuller. With it comes one of the latest runnig version of the BoA-Software, but it would be good to check for new BoA files from the CVS, or you can ask the programmers for a tarball.
[CV: Is this the first time that CVS is explicitly mentioned? I think it should be clear from the outset of the document from where the main bits of software should be obtained, regardless of which installation option is chosen (since the options are listed in the intro part of Section 3).]

3.3.1 Download the boa_build script

You can download the latest version of *boa_build* via anonymous ftp from the ftp-server of the MPIfR Bonn:

```
ftp ftp.mpifr-bonn.mpg.de (anonymous)
cd outgoing/schuller/boa
```

or using a web browser and enter as URL

```
ftp://ftp.mpifr-bonn.mpg.de/outgoing/schuller/boa/
```

If your computer is located behind a firewall remember to use a proxy in your browser settings, otherwise ftp may fail.
There are two archives, one is called allBoA.tgz, the other allBoaOff.tgz. The first one is smaller than 1MB and contains only the BoA-Software and a script which will download all additional needed software from the internet. The second one, allBoaOff.tgz, is huge, nearly 14MB, because it contains all needed software packages and does not require an internet connection. This second tar archive is the recommended one, because the versions of the software packages are consistent with the BoA-Software. If you would download all additional software packages from the internet, you may get newer version than the with BoA tested ones and may encounter problems.
[CV: The above sentence doesn't make sense.]
If you want to see and reduce some example data then you should download the archive *exFits.tgz* as well and extract it in your installation directory.
For further details see also the README file (which is also included in the archives).

3.3.2 Runnig the boa_build script

Before runnig the *boa_build* installation script you have to make sure that you fulfill following prerequisites:
First, in case of the allBoA.tgz archive, you need a connection to the internet, because all software packages are retrieved with *wget*, which you therefore will need, too. This is typically installed with every Linux-distribution. You won't need this if you use the allBoaOff.tgz archive. Second, an intel fortran compiler, named *ifc*, and the Gnu C-compiler *gcc* must be available. The Intel fortran compiler can be download from the Intel homepage and it is free for scientific and educational purpose. The Gnu C-compiler comes with Linux. Third need to have *swig* installed, the C-Python wrapper. It usually comes with the default python installation on your system.
Now go to the directory where you have stored the allBoA.tgz or the allBoaOff.tgz archive and run

```
tar -xvzf allBoA.tgz
```

or

```
tar -xvzf allBoaOff.tgz
```

In case of the allBoA.tgz archive this will create a directory called allBoA and in this the following files:

```
allBoA/BOA.MES1
allBoA/BoA.tgz
allBoA/README
allBoA/boa\_build
allBoA/exFits.tgz
allBoA/init
allBoA/p_cfitsio.tgz
allBoA/pySLALIB-0.4.tgz
```

And in case of the allBoaOff.tgz archive this will create a directory called allBoaOff and in this the following files:

```
boa_build_offline
BoaExtPkg.tgz
BoA.tgz
build.stat
ExtPkg
init
README
```

Change into the allBoA or allBoaOff directory and extract the archive *BoaExtPkg.tgz*:

```
cd allBoaOff
tar -xvzf BoaExtPkg.tgz
```

If you have also downloaded the example data *exFits.tgz* then you can now copy them here:

```
cp exFits.tgz allBoaOff/
```

Then run the script *boa_build* or *boa_build_offline*, respectively, by typing:

```
./boa_build
```

or

```
./boa_build_offline
```

You will be prompted to enter yes or no for the installation of each single software package.
If you want to intall everything you can also enter

```
./boa_build_offline -all
```

At the beginning, the script will ask you in which directory you want to install the BoA software:

```
allBoaOff> ./boa_build_offline -all
  ...boa_build> In which directory do you want to install the BoA software?
  ...boa_build> (default: /home/smueller)
```

If you don't want to use the default installation path, then please enter the complete path, e.g. */home/smueller/BoA/*, and don't forget to create this directory if not already present.
The script will create in this installation directory six sub-directories, *bin*, *BoA*, *include*, *lib*, *man* and *tmp* where all necessary files will be installed. The required disk space is about 161 MB.
In case of the on-line installation you will need for the installation of F2PY to enter the password "guest":

```
Getting and installing F2PY
Press Enter for password
Logging in to :pserver:anonymous@scipy.org:2401/home/cvsroot
CVS password: guest
```

If one or more of the servers providing the software sources is down or cannot be reached you can alternativly download all needed software packages from the LABOCA web page at the AIRUB. Just copy all archives into the directory *boa_build* and run the script *./boa_build* again.
The latest release of f2py contained a bug that prevented wrapping the BoA f95 modules. Pearu Peterson fixed it and Frank has downloaded f2py from CVS and made a tarball which is included here (*f2py2e.tar.gz*). Untar and install as usual. Note that the scipy distutils must be installed separately. **[CV: If only for my own info, what does "scipy distutils" mean? Maybe add a reference to the appropriate section here?]**
During the installation some environment variables are set. To have these available in a later session or different window you have to set these in your .cshrc, .tcshrc or .bashrc file.
Example for the .bashrc ($HOME is your home directory):

```
  # Creating an alias to start BoA
  alias boa='python -i $HOME/BoA/bin/BoA/BoaStart.py'
  # Creating the MBFITSXML environment variable
  export MBFITSXML=$HOME/BoA/bin/BoA/MBFits.xml
  # Updating variables PATH and PYTHONPATH
  export PATH=$HOME/BoA/bin:$PATH
  export newPYTHON=$HOME/BoA/lib/python2.3/site-packages
  export PYTHONPATH=$newPYTHON
  export PYTHONPATH=$newPYTHON/Numeric:$PYTHONPATH
  export LD_LIBRARY_PATH=$HOME/BoA/lib:$LD_LIBRARY_PATH
  # Defining the PGPLOT_DIR variable
  export PGPLOT_DIR=$HOME/BoA/lib/pgplot/
  export LD_LIBRARY_PATH=$PGPLOT_DIR:$LD_LIBRARY_PATH
  export PGPLOT_DEV=/XWINDOW
```

```
Example for the .cshrc and .tcshrc (\$HOME is your home directory):
```

```
  # Creating an alias to start BoA
  alias boa 'python -i $HOME/BoA/bin/BoA/BoaStart.py'
  # Creating the MBFITSXML environment variable
  setenv MBFITSXML $HOME/BoA/bin/BoA/MBFits.xml
  # Updating variables PATH and PYTHONPATH
  setenv PATH $HOME/BoA/bin:$PATH
  setenv newPYTHON $HOME/BoA/lib/python2.3/site-packages
  setenv PYTHONPATH $newPYTHON
  setenv PYTHONPATH $newPYTHON/Numeric:$PYTHONPATH
  if ($?LD_LIBRARY_PATH) then
    setenv LD_LIBRARY_PATH $HOME/BoA/lib:$LD_LIBRARY_PATH
```

```
  else
    setenv LD_LIBRARY_PATH $HOME/BoA/lib
  endif
  # Defining the PGPLOT_DIR variable
  setenv PGPLOT_DIR $HOME/BoA/lib/pgplot/
  setenv PGPLOT_DEV /XWINDOW
```

Here is what Frank actually put into his .cshrc :

```
#================================================================
# System variables, paths, and alias needed for python and BoA
#
if (-e /opt/intel/compiler60/ia32/bin/iccvars.csh) then
  source /opt/intel/compiler60/ia32/bin/iccvars.csh
endif
#
set boahome = '/aux/pc191b/bertoldi/boa'
#----------------------------------------------------------------
setenv PYTHONPATH .:/opt:/opt/intel/compiler60/ia32/lib/
setenv PYTHONPATH {$PYTHONPATH}:/usr/lib
setenv PYTHONPATH {$PYTHONPATH}:$boahome/lib/python2.3
setenv PYTHONPATH {$PYTHONPATH}:$boahome/lib/python2.3/site-packages
setenv PYTHONPATH {$PYTHONPATH}:$boahome/lib/python2.3/site-packages/Numeric
# ---------------------------------------------------------------
alias  boa "python -i BoaStart.py"
alias  cleanboa "/bin/rm -f BOA.MES* *.pyc"
setenv MBFITSXML $boahome/openboa/BoA/MBFits.xml
setenv PGPLOT_DIR $boahome/lib/pgplot/
setenv LD_LIBRARY_PATH {$PGPLOT_DIR}:{$LD_LIBRARY_PATH}
set path=( $boahome/bin $path )
# ---------------------------------------------------------------
setenv PYTHONSTARTUP /homes/dmuders/bin.python/interactive.py
setenv PGPLOT_DEV    /xserve
# ---------------------------------------------------------------
setenv CVSROOT :ext:openboa@glx-brazil.ias.u-psud.fr:/home/openboa/
setenv CVS_RSH ssh
alias  cvsupdate "cvs -q update -d"
alias  cvscommit "cvs commit -m "
# ---------------------------------------------------------------
```

The default ppgplot installation will have numarray support, but currently BoA uses Numeric. You will need to reinstall ppgplot now, else BoA will fail! To do this, enter allBoA/ppgplot-1.3 and edit the file setup.py such that it looks like:

```
...
#from numarray.numarrayext import NumarrayExtension
# comment-out the previous line ("from numarray ..."),
# and comment-in the following to force the use of "Numeric"
raise ImportError
...
```

Then from the ppgplot-1.3 directory, clean the installation:

```
make clean
```

and reinstall:

```
python setup.py install --prefix=/aux/pc191b/bertoldi/boa
```

where you put in the path you installed to earlier.

3.3.3 Known problems:

It may happen, that the libpgplot.so and libcpgplot.so libraries will not be found:

```
/home/smueller> boa
Traceback (most recent call last):
  File "/home/smueller/BoA/BoA/BoaStart.py", line 26, in ?
    import ppgplot
  File "/home/smueller/BoA//lib/python2.3/site-packages/ppgplot/__init__.py", line 1, in ?
    from _ppgplot import *
ImportError: libpgplot.so: cannot open shared object file: No such file or directory
```

For a first easy solution and if you have root privileges then you can just link or copy the files *libcpgplot.a*, *libcpgplot.so*, *libpgplot.a* and *libpgplot.so* from $home/BoA/lib/pgplot/ directory to the local lib-directory, i.e. /lib/ or /usr/lib/.
Another possible error message can occur, if swig is missing:

```
  File "/home/USERNAME/BoA/BoA/cfitsio.py", line 4, in ?
     import _cfitsio
  ImportError: No module named _cfitsio
```

Here the problem is that swig was not installed on the computer. You also need swig to compile p_cfitsio (download it from http://www.swig.org/).

3.4 Manual installation of each package as user

This subsection describes what software ist needed to run BoA and how to install it on an Intel-Linux-PC as user without root-privileges. The only package which has to be installed by root is the Intel-Fortran compiler.

3.4.1 Python 2.3.2

Download the file Python-2.3.2.tgz from the Python homepage:
(http://www.python.org/2.3.2/)

Execute the following commands:

- `tar -xvzf Python-2.3.2.tgz`
- `rm Python-2.3.2.tgz`
- `cd Python-2.3.2/`
- `./configure --prefix=/DESTINATION-PATH/ --exec-prefix=/DESTINATION-PATH/`
  e.g.: `./configure --prefix=/home/smueller/LABOCA/Python/ --exec-prefix=/home/smueller/LABOCA/Python/`
- `make`
- `make install`
- `make clean`
- `cd ..`
- `rm -rf Python-2.3.2`

Python is then installed into /DESTINATION-PATH/bin/

Add the PATH for Python to the environment variable, e.g. for the tcsh edit the .cshrc file in the home directory and add the following line:
`setenv PATH /DESTINATION-PATH/bin:$PATH`
e.g.: `setenv PATH /home/smueller/LABOCA/Python/bin:$PATH`

Or you can define an alias:
`alias python2.3.2 '/DESTINATION-PATH/bin/python'`
e.g.: `alias python2.3.2 '/home/smueller/LABOCA/Python/bin/python'`

Then run the command `source /home/USERNAME/.cshrc` to activate your changes or open a new terminal window.
The time needed to install Python 2.3.2 on a PIII 500/,MHz with 128/,MB RAM was aproximately 20/,min.

3.4.2 Numeric:

(ATTENTION: Numeric is kind of obsolete, from now on only using numarray, see below.)
Download the file Numeric-23.1.tar.gz from the numeric/numpy homepage (http://www.pfdubois.com/numpy/)

Execute the following commands:

- tar -xvzf Numeric-23.1.tar.gz
- rm Numeric-23.1.tar.gz
- cd Numeric-23.1/
- python setup.py install
  or: python2.3.2 setup.py install
- sh makeclean.sh

Numeric can then be found at
/DESTINATION-PATH/include/python2.3/Numeric
and
/DESTINATION-PATH/lib/python2.3/site-packages/

ATTENTION: Don't use the Numeric-22.0-1.i386.rpm, it causes some problems with mbfitsEntity.py.

3.4.3 numarray

Download the file numarray-0.9.tar.gz from the sourceforge.net homepage.

Execute the following commands:

- tar -xvzf numarray-0.9.tar.gz
- rm numarray-0.9.tar.gz
- cd numarray-0.9/

- python setup.py install
  or: python2.3.2 setup.py install

numarray can then be found at
/DESTINATION-PATH/include/python2.3/numarray
and
/DESTINATION-PATH/lib/python2.3/site-packages/numarray

3.4.4 pgplot

Installation of pgplot (http://www.astro.caltech.edu/ tjp/pgplot/)

You can download the source code file pgplot5.2.tar.gz from ftp://ftp.astro.caltech.edu/pub/pgplot/pgplot5.2.tar.gz

Execute the following commands:

- cd /usr/local/src/
- cp /pgplot5.2.tar.gz .
- gunzip -c pgplot5.2.tar.gz — tar xvof -
- rm pgplot5.2.tar.gz
- mkdir /usr/local/pgplot
- cd /usr/local/pgplot/
- cp /usr/local/src/pgplot/drivers.list .
- vi drivers.list

(De)select the (un)wished drivers with the # sysmnol or download testet list from http://www.openboa.de/pgplot/drivers.list. [2]

- /usr/local/src/pgplot/makemake /usr/local/src/pgplot/ linux g77_gcc
- make
- make clean

[2] http://www.openboa.de/pgplot/drivers.list

Usefull but not needed for pgplot:

- make cpg
- ld -shared -o libcpgplot.so –whole-archive libcpgplot.a
- cp /usr/local/pgplot/libpgplot.so /usr/lib/

Set the environment variables:

UNIX csh: setenv PGPLOT_DIR /usr/local/pgplot/
UNIX sh: PGPLOT_DIR="/usr/local/pgplot/ls"; export PGPLOT_DIR put this for example into /etc/bashrc (root rights required) or as user put it into your .cshrc or .bashrc files

More information can be found on the following website: http://www.astro.caltech.edu/ tjp/pgplot/faq.html

3.4.5 ppgplot

ATTENTION: There is a new version of ppgplot available at:
http://efault.net/npat/hacks/ppgplot/

Download the latest version from the website mentioned above, i.e.:
http://efault.net/npat/hacks/ppgplot/dist/ppgplot-1.3.tar.gz

After downloading execute the following commands:

- tar -xvzf ppgplot-1.3.tar.gz
- cd ppgplot-1.3
- python setup.py install
  or: python2.3.2 setup.py install

Problems:

- ATTENTTION:

  ppgplot can be compiled either with numarray or Numeric support, but not both!

- There can be a problem plotting with ppgplot, with the earlier version 1.1 and also with the version 1.3.

  The system this problem occured was a SuSE 7.3 on a notebook, python 2.3.3 (also tried 2.2.1) with Numeric 23.1 (21.3) and numarray 0.8.

  In essence, whenever trying to plot a Numeric array, the following error occured: "ppgplot.typeerror: cannot cast vector to floats" (in the 1.1 version)

  Also trying to to run one of the examples with version 1.3 resulted into an error:

  File "ex_graph.py", line 24, in ? ppgplot.pgpt(xs,ys,9) _ppgplot.typeerror: object is not an array

  It is the Numeric arrays that cause the problem, the numarray ones are ok, the standard example na_ex_graph.py plots fine.

  The tested example was:

  >>> X = [1.,2.,3.,4.,5.,6.,7.,8.,9.,10.]

  >>> dataX = Numeric.reshape(X,(Numeric.shape(X)))

  >>> dataX array([1., 2., 3., 4., 5., 6., 7., 8., 9., 10.])

  >>> ppgplot.pgbox('BCTSN',0.0,0,'BCTSNV',0.0,0)

  >>> ppgplot.pgline(dataX,dataX)

  Traceback (most recent call last):

  File "<stdin>", line 1, in ?

  _ppgplot.typeerror: object is not an array

  In the earlier version of ppgplot the message was "ppgplot.typeerror: cannot cast vector to floats".

  Commenting out the numarray support in setup.py does not change the error.

  A typical error message when trying to plot can look like the following example:

```
Traceback (most recent call last):
File "<stdin>", line 1, in ?
File "/home/smueller/bin/BoA/BoaFocus.py", line 127, in scanFocus
self.BoaB.Graphic.Focus.scanFocus()
File "/home/smueller/bin/BoA/BogliFocus.py", line 65, in scanFocus
self.plotDataXY(dataX,dataY,'p',k)
File "/home/smueller/bin/BoA/BogliPlot.py", line 337, in plotDataXY
ppgplot.pgpt(dataX,dataY,self.symbolPoint) # plot points
```

```
_ppgplot.typeerror: object is not an array
```

Solution:

The problem was, that "ppgplot" was compiled with numarray support, and then it was tried to use it with Numeric. This does not work! That is: the ppgplot built for numarray, and the ppgplot built for Numeric are *not*, so to say, compatible. You have to select which python extension you will support (numarray, or Numeric) when you *build* ppgplot. You can't have both at the same time.

So, in order to make the example above work, try this:

1. Clean up everything in the source distribution (e.g. do a "make clean", or remove the distribution, and unpack it from scratch)

2. Edit "setup.py". Comment-OUT the line:

from numarray.numarrayext import NumarrayExtension

AND ALSO comment-IN the line (a few lines below):

raise ImportError

3. Rebuild and install everything as described in INSTALL.

The resulting version of ppgplot will be able to work with Numeric arrays, but *not* with numarray arrays.

Instead of point 3 above, you can try saying "make" which will build and install everything *in-place*. Then you can run python, *without changing the current directory*, and try the example above. This will help you determine if there is some other incompatible ppgplot version somewhere in you path.

When you build "ppgplot", please pay attention to the output. You should see a line identifying which ppgplot version (the numarray or the Numeric) you are building:

using Numeric...

or

using numarray...

3.4.6 Installation of the Intel Fortran Compiler

You can download the latest version of the Intel Fortran Compiler from the Intel homepage [3] or [4] and in case of non commercial, educational or scientific purpose you will get a license key for free. Tested versions are ifc 7.8, ifc 8.0 and ifc 8.1.

[3] http://www.intel.com/software/products/compilers/flin/
[4] http://www.intel.com/software/products/compilers/flin/noncom.htm

In the following the installation of the ifc 8.0 is described as an example. After downloading the tar archive *l_fc_p_8.0.034.tar.gz* you have to extract it via

```
tar -xvzf l_fc_p_8.0.034.tar.gz
```

The change into the directory *l_fc_p_8.0.034/*

```
cd l_fc_p_8.0.034/
```

and execute the *install.sh* command

```
./install.sh
```

You will be prompted to enter a number according to the kind of machine type, Kernel version and glibc version you are using. You will have to enter where a valid FLEXlm license can be found in your system (you get the license from intel for educational or scientific purpose for free). You can create e.g. a directory */opt/intel_fc_80/licenses* and put there the file *l_for_61850218.lic*.
To use the intel fortran compiler *ifc* you will have to include the path */opt/intel_fc_80/lib/* to your *LD_LIBRARY_PATH* variable and the path */opt/intel_fc_80/bin/* to your local *PATH* variable, e.g.:

```
# ---- for fortran
setenv LD_LIBRARY_PATH /lib:/usr/lib:/opt/intel_fc_80/lib/
setenv PATH /opt/intel_fc_80/bin/:/opt/intel_fc_80/lib/:$PATH
```

3.4.7 swig

Now you have to install the Swig C-Python wrapper [5] which you can download for free. The tested version is swig 1.3.19.
After downloading, run:

- ./configure –prefix=/home/USERNAME/DESTINATION-PATH/
- or e.g.: ./configure –prefix=/home/USERNAME/BoA/
- make
- make install

[5] http://www.swig.org/

3.4.8 pySLALIB

Installation of the pySLALIB library (you need a directory *lib* in your home directory):

- cp pySLALIB-0.4.tgz $HOME/lib/
- cd $HOME/lib/
- tar xzf pySLALIB-0.4.tgz
- tar xzf slalib.tgz
- rm pySLALIB-0.4.tgz slalib.tgz
- cd slalib
- ./compile
- cd ../pySLALIB-0.4
- ./compile
- cp slalibmodule.so $HOME/lib/python2.3/site-packages/
- or: /DESTINATION-PATH/lib/python2.3/site-packages/

3.4.9 cfitsIO

Installation of the fits package CFITSIO:
Tested version was cfitsio2440b.tar.gz

Installation:

You have to be root!
The CFITSIO library is built on Unix systems by typing:

- ./configure [–prefix=/target/installation/path]
- make (or 'make shared')
- make install (this step is optional)

3.4.10 Scipy Distutils

There is a new version of the scipy distutils available: scipy_distutils-latest.tar.gz
These must be installed separately. Untar and run

```
python setup.py install  --prefix=$outDir
python setup_scipy_distutils.py install --prefix=$outDir
```

This should result in the appearance of a directory

```
$outDir/lib/python2.3/site-packages/scipy_distutils
```

3.4.11 pcfitsIO

Installation of the fits package CFITSIO:

3.4.12 Path structure

In the following the entire path strucure setup in the system file .cshrc is shown as an example:

```
setenv PYTHONPATH ".:
        /opt/intel/compiler60/ia32/lib/:
        /usr/local/lib/python2.3:
        /usr/local/lib/python2.3/site-packages:
        /home/bertoldi/bin:
        /opt:
        /usr/lib"

setenv PYTHONSTARTUP /home/bertoldi/bin/pythonstartup.py
setenv MBFITSXML  ./MBFits.xml
setenv PGPLOT_DIR /usr/local/pgplot/
setenv PGPLOT_DEV '/XSERVE'
setenv CVSROOT :ext:openboa@glx-brazil.ias.u-psud.fr:/home/openboa/
setenv CVS_RSH ssh

if (-e /opt/intel/compiler60/ia32/bin/ifcvars.sh) then
  source /opt/intel/compiler60/ia32/bin/ifcvars.csh
endif

if ($?LD_LIBRARY_PATH) then
  setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:/lib:/usr/lib
```

```
    else
      setenv LD_LIBRARY_PATH /lib:/usr/lib
endif
```

3.4.13 Listing of python site-packages

In the following the comlete list of site-packages of the python site-packages directory is shown:

```
/usr/local/lib/python2.3/site-packages
<site-packages> ls -Floh
total 2.9M
drwxr-xr-x    5 root         4.0k Feb 28 10:18 Numeric/
-rw-r--r--    1 root            8 Feb 28 10:18 Numeric.pth
-rw-r--r--    1 root          119 Feb 21 22:57 README
drwxr-xr-x   13 root         4.0k Feb 22 04:09 Scientific/
drwxr-xr-x   10 root         4.0k Feb 22 04:14 _xmlplus/
lrwxrwxrwx    1 root           11 Feb 26 19:40 f2py2e -> f2py2e-2.39/
drwxr-xr-x    3 root         4.0k Feb 26 19:39 f2py2e-2.37/
drwxr-xr-x    3 root         4.0k Mar  9 13:59 f2py2e-2.39/
-rw-r--r--    1 root          11k Feb 22 04:56 fitsio.py
drwxr-xr-x   10 root         4.0k Feb 22 04:13 numarray/
-rwxr-xr-x    1 root         1.2M Feb 22 04:05 pcfitsio.so*
-rwxr-xr-x    1 root         1.2M Feb 22 04:05 pcfitsio.so.backup*
drwxr-xr-x    2 root         4.0k Feb 28 15:49 ppgplot/
drwxr-xr-x    3 root         4.0k Feb 22 04:03 scipy_distutils/
-rwxr-xr-x    1 root         423k Feb 22 04:06 slalibmodule.so*
```

3.4.14 Readline interactive tool

The readline library we are right now providing with our toolset comes with every Linux distribition. The user has to have the package *readline-devel* installed to make the python build procedure recognize the readline interactive stuff.

For reference, please see:

www.pbone.net → search for rpms containing a specific file → enter libreadline.so → Press search button.

3.4.15 Additional Libraries

There are many additional libraries needed. Some of them have to be linked like the following example:

```
in /usr/lib check if liblapack.so is there and create symbolic links
using the following commands:

ln -s liblapack.so liblapack.so.3
ln -s liblapack.so.3 liblapack.so.3.0.0

and install the libblas library (blas=Basic Linear Algebra Subprograms)

there are rpm available at http://rpmfind.net/ just search for libblas.so

as root do: rpm -i blas-3.0-18.i386.rpm

lrwxrwxrwx 1 root     root    14 Feb 16 17:55 liblapack.so -> liblapack.so.3
lrwxrwxrwx 1 root     root    18 Feb 16 17:54 liblapack.so.3 -> liblapack.so.3.0.0
-rwxr-xr-x 1 bertoldi users   4M Feb 16 17:51 liblapack.so.3.0.0

lrwxrwxrwx 1 root     root    12 Feb 16 17:58 libblas.so -> libblas.so.3
lrwxrwxrwx 1 root     root    16 Feb 16 17:58 libblas.so.3 -> libblas.so.3.0.0
-rwxr-xr-x 1 bertoldi users 335k Feb 16 17:56 libblas.so.3.0.0
```

3.5 Updating BoA

Before you start boa, we advise you set up your .cshrc or so properly and start a new shell. Then change to the *BoA/fortran/* directory and compile the fortran module according to the instruction given in the BoaData.f90 header. Currently that means:

```
cd BoA/fortran/
ifc  -c -w svd.f90
f2py -c --fcompiler=intel -m f90 BoaData.f90 BoaF1.f90 \
    BoaFortranBaseLine.f90 BoaChanAna.f90 BoaSNF.f90 svd.o
```

Or you can run a little script, located in the *BoA/fortran* directory, called *doF2py*, by changing into this directory and typing

```
cd BoA/fortran/
./doF2py
```

There will be lots of warnings in the f2py, but in the end you should see a link line, and no report on errors. Then you have created a so called extention module f90.so. With this you can now start boa in the BoA directory. Good luck!
If you did not use the installation script *boa_build* or if you want to update BoA to a newer version you simply can do this by copying all BoA routines into the BoA directory. The lastest version of all BoA-routines can be found at http://www.astro.rub.de/laboca/download/BoA.tgz
[**CV: It is not clear to me here what this url is for. It's fine to give this info here, but much earlier on in Section 3 it should be clear from where the various info/software etc should be downloaded/obtained. Since this link is just a file download the reader is sort of blind as to whether this is what they are looking for or not...unless things are clarified earlier on.**]
Using the installation script *boa_build* of Frederic Schuller and $HOME/BoA/ as installation directory the BoA directory can be found at $HOME/BoA/bin/BoA/ . If you have downloaded a newer BoA version and copied everything to the BoA directory you will have to recompile the fortran routines in $HOME/BoA/bin/BoA/fortran/ which is described above or in chapter 7.4.

4 Data organisation

4.1 Input data

The data acquired at the APEX telescope are stored in a new file format, known as the MB-Fits format (for Multi-Beam FITS format, see Hatchell et al.) These files contain:

- the raw data as provided by the Frontend-Backend in use at the telescope
- data associated parameters: time of the observations, positions on the sky...
- a description of the complete Scan (eg. for a map: number of lines, steps between lines...)
- parameters of the receiver channels in the array: relative positions, relative gains

For the development of BoA, as long as no MB-FITS file with real observed data is available, we made use of Mambo files converted to the MB-FITS format. A dedicated Python module was written to read the content of a FITS file containing MAMBO data, and create a file with the same data plus the appropriate data associated parameters in the MB-FITS format. This module is able to convert any kind of MAMBO observations: Pointing, Focus, Skydip, On-Off, Raster map, as well as maps performed in the Fast Scanning mode. The files generated in the MB-FITS format can then be read into BoA with the appropriate method (see Section...).
[**CV: Cat: Is there a Section ref available yet?**]

4.2 Data objects

The manipulation of data within BoA is done with objects of the DataEntity class (see also Section **??**). One object of this class is stored in the currData attribute of the BoaB object; this is the current object, on which all reduction procedures can be applied. Additional objects of this class can be created by the user.
The definition of DataEntity objects enables to store the raw data and the reduced data, as well as associated parameters, as described below.

4.2.1 Raw data

When a file in the MB-Fits format is read into BoA, the data from a complete scan are stored in the following attributes of the current DataEntity object:

- FITS_Header: this is a dictionary, which contains some general descriptive parameters for the scan. This includes: telescope name and location, scan type and global parameters, wobbler mode and parameters, source name, coordinates and date of observation, frontend-backend combination in use.

- Data_Obs: the raw bolometer data (or counts) are stored in a 2-dimensional array (integrations × bolometers).

- Array parameters: the receiver channel parameters are stored in two attributes: Array_Gain (relative gains), and Array_Geo (offsets with respect to the reference bolometer).

- LST: local apparent sidereal time at integration midpoint.

- Az, El: azimuth and elevation actually observed, including pointing corrections and wobbler offsets.

- Lon, Lat: longitude and latitude offsets from the source in user native frame (intermediate coordinates).

- Wobbler_Sta: indicates the wobbler status for each integration point. In case of wobbler switching observations, this is a succession of 'ON' and 'OFF'.

- Subscan_Num, Subscan_Epo, Subscan_Time: when a scan is composed of several observations (or subscans), these arrays contain, respectively: an increasing sequence of integers, the calendar date (floating value in years) of the observation start, and the local sidereal time (in seconds) at observation start.

- Track_Az, Track_El: tracking errors in Azimuth and Elevation *(not implemented yet)*.

4.2.2 Derived data

As soon as the raw data are read, some additional quantities are derived and stored in the following attributes:

- Channel_Sep: separation between bolometers, computed from Array_Geo

- UT: universal time at each integration *(not implemented yet)*.

- phase differences: for observations performed with wobbler switching, pairs of ON–OFF integrations are extracted from the Wobbler_Sta attribute, and the phase differences can be computed. The results are stored in the 3-dimensional arrays Data_Obs_p, Data_Bac_p, Data_Red_p, Data_Noi_p, Data_Flag_p, where the 1st dimension corresponds to integrations, the 2nd dimension means: 0 = phase diff, 1 = phase1, 2 = phase2, and the 3rd dimension corresponds to pixels. The following parameters are also computed, where the values associated with the phase difference is computed as the mean value of phase1 and phase2: LST_p, Az_p, El_p, Lon_p, Lat_p, UT_p.

4.2.3 Processed data

- Data_Red: this 2D array contains the reduced data after each processing step. It is first initialised with a copy of the raw data.
- Data_Bac: after every processing step, the data previously contained in Data_Red are copied to this backup array, to enable a quick 'undo' procedure.
- Data_Flag: used to flag inidividual data points.
- Data_Noi: rms noise associated with individual data points.
- Channel_Flag: used to flag channels.
- FFCF_CN: correlated noise.
- FFCF_Gain: computed flat field.
- Channel_rms: rms noise, per channel.
- Channel_mean: mean value, per channel.
- Results: depending on the type of observation, this can contain one or several numbers. It can also contain the results of several scans of the same type (e.g. when one wants to combine the results of several successive pointings).
- Weights: to be applied to the results (see previous item).
- Command_History: this is used to store the commands that are typed into BoA, in string format. This can then be saved into a script file, that can later be edited and imported again into BoA.

4.2.4 BoA Status

In addition to the raw, derived, processed data and associated parameters, a dictionary called Status_Dic is used to store some parameters related with the reduction steps that have been performed with BoA. This dictionary contains the following keys:

- Gain_Ele_Cor_Done
- Baseline_Cor_Ord
- Noi_Cor_Done
- Opa_Cor_Done
- Flux_Cal_Done
- Pha_Dif_Done

4.3 Output data

With the aim of comparing the results of a data reduction performed with BoA with those obtained with existing packages (NIC, MOPSI), a procedure similar to the one described in Section 4.1 was written to convert an MB-FITS file to a FITS file with the same format as for MAMBO-ABBA data. These two modules were tested by comparing the results of the reduction performed with NIC or MOPSI on the original data file on one hand, and on the file converted to the MB-FITS format, then back to the MAMBO format on the other hand. These tests were successful (no difference can be seen in the results obtained with and without conversions) for the following software/observation type combinations:

- Pointing / NIC
- Focus / NIC
- Skydip / NIC
- On-Off / NIC
- Raster map / MOPSI
- Fast Scanning map / MOPSI

5 Usage

[CV: General comment: Somewhere in this chapter we should mention that graphics commands (i.e. Bogli) are discussed in the next Chapter.]

5.1 Starting BoA

[CV: This section doesn't seem clear to me. What about the alias we created in Section 3, and what is meant by "apex-Calibrator"? I think it's important that this section is very clear, and at the moment it seems to require the reader to have some implicit knowledge...]
First you have to define an environment variable MBFITSXML, whose value is the path to the file MBFits.xml, including the file name itself (e.g.: setenv MBFITSXML ./MBFits.xml)
To invoke BoA call python in interactive mode (*-i*) with the file *BoaStart.py*:

```
python -i BoaStart.py
```

Alternatively, from an already running python session (e.g. within a shell window running the apex-Calibrator), it is possible to import the BoA functionalities and commands by giving to the python prompt:

```
>>> execfile('BoaStart.py')
```

BoA then prints a welcome message providing version information and changes the prompt. Nevertheless, you are still in the interactive python layer. The start script *BoaStart.py* imports a set of modules, instantiates the most essential objects and makes the respective methods available.
In particular, it instantiates an object of the Map class, called *data*, to which all BoA commands described in the remaining of this section are applied. It is nevertheless possible to instantiate several data objects within one BoA session; then, applying BoA methods to a data object with a different name that *data* requires to enter the full syntax, including the full name of the method, as opposed to the shortcuts explained below.

5.2 Description of how Boa works

[CV: Need a proper title but this is the general idea.]

5.2.1 Methods

The functionalities of BoA are accessed by directly calling the appropriate methods from the interactive python layer. This ensures the full availability of all python and ppgplot facilities. As the method names to be called from the python layer may be rather long, the start script *BoaStart.py* provides a set of convenient abbreviations for those methods which are meant to be called from the user ("public" methods).

Example: The name of the method to open a new graphic device is *DeviceHandler.openDev* and can be called by

```
DeviceHandler.openDev()
```

[CV: Should specify here: "in Bogli terms, see Chapter 6, or in terms of Boa commands...open() etc.]
or more comfortably by

```
open()  or  ope()  or  op()
```

where the parentheses are mandatory.
Python ensures no real difference between private and public attributes. There are only hidden attributes but this hiding can be overcome easily. Therefore the user might set any attribute directly and call any method. This is not advisable and may easily corrupt the whole BoA session. It is more recommendable to just use those methods for which the start script *BoaStart.py* provides abbreviations. In the following these are called user methods. A detailed description of all user methods, their arguments, output and abbreviations is given in section 5.3.
[CV: Does Section 5.3 really include "all" user methods? It doesn't, since it doesn't include things like plot and multiplot, but at the same time it is not simply restricted to Boa commands. In truth, Section 5.3 and also Table 4 include all the Boa commands and some Bogli-only commands. We have to somehow make this clearer.]

5.2.2 Arguments

Nearly all user methods require arguments to be passed. Nevertheless, the methods provide default arguments which thus may be omitted. In this case many methods just supply status information.
Example: The method *indir()* sets the desired input directory and requires the name as argument:

```
indir(''/home/user/data/'')
```

The directory name is a string argument and has to be passed embedded in double or single quotes.
[CV: Even though can use single or double quotes it might be clearer to just stick to one or the other consistently in the examples throughout this document.]
Omitting the argument does not change the input directory and results in the supply of the current directory name:

```
indir()
```

In case an argument has to be typed more often a python variable might be used:

```
a='/home/user/data/'
indir(a)
```

Some methods require a list as argument. In python a list is embedded in square brackets with a comma as separator. Python provides a variety of functionalities to manipulate lists.
Example: The method *signal()* plots the time series of the data (flux density or counts versus time). **It allows the user to define the list of channels plotted: [CV: Note:- re-worded sentence above.]**

```
signal([18,19,20])
```

To create a list you might use the python function *range()*:

```
mylist=range(1,163)
signal(mylist)
```

or in one line:

```
signal(range(1,163))
```

Even if the list contains only one element the square brackets are mandatory:

```
signal([5])
```

User methods can also be called using keyword arguments of the form *keyword = value*.
Example: By default, the *signal()* method plots the signal versus time connecting the datapoints with lines:

```
signal()
```

If you prefer to see the individual datapoints without lines, you can modify the value of the *style* argument:

```
signal(style='p')
```

5.2.3 Output

Most user methods supply status information as screen output when being called. The amount of information displayed can be restricted using the message handler associated with the main *data* object:

```
data.MessHand.setMaxWeight(4)
```

where the arguement is an integer value between 1 and 5, with the following meaning:

- 1: errors, queries
- 2: warnings
- 3: short info
- 4: extended info
- 5: debug

5.3 Detailed description of all user methods

[CV: I think switching between referring to "methods" and "commands" is confusing. Should be consistent.
]
[CV: Also could probably do with a better title here, but i haven't thought of one yet.
]
[CV: I think at the moment this section seems a bit confused...I've tried to "re-shuffle" the list of commands so they are roughly grouped in terms of what they do...it would maybe be an idea to even make separate sections for these groupings....from my point of view they seem to be: 1)data reduction commands, 2) file-reading, 3) display device setup (which are Boa-independant Bogli commands and so are described in detail in Chapter 6), and finally 4) plotting commands (which are related to Boa and so are not described in Chapter 6).
]
In this section you will find detailed descriptions of all user methods, their arguments, output and abbreviations and some examples for the different tasks possible to execute in BoA. Many commands have an abbreviated form, and these are listed in Section 5.6. Just enter them at the boa> prompt.
[CV: reduce, as below, gives error "expected 2 arguments, got 1"
]

5.3.1 All at once: the *reduce* method

[CV: What is meant by "all at once"? Data-reduction I guess...title needs clarifying.]
Reading and processing a file can be done in one single step, using the *reduce* method. This method takes the name of an MB-Fits file as argument, e.g.:

```
reduce('APEX-600')
```

The extension (*.fits* or *.fits.gz*) may be omitted. This method first reads the file, then opens a graphic device if no device has previously been openned. Then, according to the type of observation found in the file, it calls the relevant methods to process the data.

If a file has already been partially read, this method will only read the subscans not already present in the *data* object and then process the full set of data. This can be useful e.g. for an iterative real time display of raster maps, where the map is displayed after completion of each subscan.

5.3.2 Pointing

Processing a Pointing scan requires an object of the class Point. Since the default *data* object is of class Map, it has to be redefined before reading the file. Then the method to process the data is called *solvePointing*. Optionally, the method *showPointing* can be called to show the results on a map:

```
data = BoaPointing.Point()  # instantiate a Point object
data.read('APEX-600')       # fill it with data
data.solvePointing()        # compute pointing offsets
data.showPointing()         # display map and fitted 2D-Gaussian
```

5.3.3 Focus

The recommended way to conduct laboca focus observations is to perform a series of n*3 short, symmetric on-offs, e.g. 3 or 6*(4*5sec). For this simply the onoff has to be reduced and then the results can be fitted by a parabola.

5.3.4 Skydip

[CV: Anything to add here yet?]

5.3.5 OnOff

[CV: Anything to add here yet?]

5.3.6 Mapping

Several methods are provided to construct a map, taking into account the relative positions of the bolometers in the instrument. The *slowMap* method computes exact positions and loops over the pixels of the resulting map to calculate the contributions to the flux at a given position from all bolometers. This is a *very* slow method.
The *fastMap* method loops over the signal series in each bolometer, and dumps fluxes at the nearest pixel on the final map. Then the maps produced from each bolometer are coadded. This method makes use of operations on arrays, and is thus very fast.

```
read('lissajou')
open()                  # open an XWindow device; see ~\ref{openplotCB.sub} below
data.fastMap()       # reconstruct a map with the fast method
```

5.3.7 Beam maps

[CV: Anything to add here yet?]

5.3.8 Reading a FITS file

Reading a FITS file into BoA is done with the *read()* command. You may want to define the input directory first:

```
indir("../Fits/")      # set the input directory
read('APEX-600')       # read file APEX-600.fits
```

[CV: Shouldn't the above "fits" be with lower case "F"?] The data are then stored in the default *data* object. It is possible to use several data objects, and to store the content of a file to a user defined object requires the following syntax:

```
data2 = BoaMapping.Map()  # define a second data object of class Map
data2.read('APEX-600')
```

5.3.9 Opening a plot window

[CV: Graphics-related commands are discussed in detail in Chapter 6, and here we give just an overview where appropriate. We should say this here.]
Opening a graphic device is done with the *open()* command:

```
open()    # open a device, default: XWindow
op()      # alternatively, use the abbreviated command (see~\ref{abbreviationsCB.sub})
```

The default is to open an XWindow. You can use:

```
op('?')
```

to get a list of all recognized devices. Alternatively, if you know which device you want you can enter it directly, for example:

```
op('/ps')
```

You can also open a named PostScript file, here a colour PostScript file named *signal.ps*, with:

```
op('signal.ps/CPS')
```

5.3.10 Clearing a plot window

[CV: Graphics-related commands are discussed in detail in Chapter 6, and here we give just an overview where appropriate. We should say this here.]
Clearing a plotting window is done with the *clear()* command:

```
clear()              # clear the active device
```

However, any plot command will first clear the active device before plotting a new graph, unless the *overplot=1* keyword is supplied.

5.3.11 Closing a plot window

[CV: Graphics-related commands are discussed in detail in Chapter 6, and here we give just an overview where appropriate. We should say this here.
]
Closing a graphic device is done with the *close()* command:

```
close()    # open a device, default: XWindow
```

5.3.12 Channel Maps

If you want to display channel maps you can do this with the command *chanmap()*. The default is to plot channel maps for all available channels. You can also specify a list of channels to be plotted.

```
read('3543')
op()                 # open an XWindow device
chanmap()            # produce channel maps for all channels
chanmap(range(26))   # channel maps for the first 25 channels
chanmap([1,4,20,55]) # channel maps for a selection of channels
```

5.3.13 azimuth

Usage: **azimuth(***optional arguments***)**

Optional arguments:

flag: flag to be used (default = 0: all valid data; -1: plot all)

limitsX

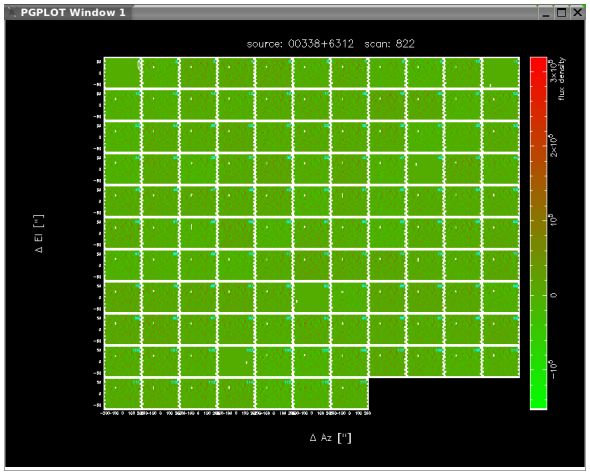

Figure 5.1: Default graphical outputs of a channel map of the source 00388+6312, including a wedge.

limitsY

style

ci

overplot

[CV: We should add a ref link to somewhere where we say what these plotting options, e.g. flag, ci, are...some are in 6.9 at the moment, but currently not all these are documented.
]
Plot the time series of the azimuth i.e. azimuth versus LST.

Example:

azimuth(style='p', ci=2, limitsY=[-14,-13])

Plot azimuth versus LST but show individual plotted points (rather than lines), make plotted points red, and only plot azimuth (y axis) from -14 to -13 degrees.

5.3.14 elevation

Usage: **elevation**(*optional arguments*)

Optional arguments:

flag: flag to be used (default = 0: all valid data; -1: plot all)

limitsX

limitsY

style

ci

overplot

[CV: We should add a ref link to somewhere where we say what these plotting options, e.g. flag, ci, are.
]
Plot the time series of the elevation i.e. elevation versus LST.

Example: as for *azimuth*, above.

5.3.15 azel

Usage: **azel(***optional arguments***)**

Optional arguments:

flag: flag to be used (default = 0: all valid data; -1: plot all)

limitsX

limitsY

style

ci

overplot

[CV: We should add a ref link to somewhere where we say what these plotting options, e.g. flag, ci, are.
]
Plot elevation versus azimuth.

Example: as for *azimuth*, above.

5.3.16 channels

Usage: **channels(***optional argument***)**

Optional argument:

chanList: list of channel numbers, of the form: [1,2,3]

'all'... 'al'...'a'

'?'

Select a channel or a list of channels to be plotted. The list is automatically sorted.

Examples:

channels([1,2,3]): list of channels to be plotted

channels(chanList=[1,2,3]): list of channels to be plotted

channels('all'): set current list to all possible channels

channels('?'): get current list of channels (the default if no argument is specified)

[CV: I've removed "plot limits" and "plot default", since these seem to be obsolete]

5.3.17 signal

Usage: **signal(***optional argument***)**

Optional argument:

chanList: list of channels, of the form [1,2,3]

flag: flag to be used

mjd: if set, use mjd instead of lst

limitsX

limitsY

style

ci

overplot

Plot the time series of the flux density i.e. flux density versus LST.

Examples:

signal(chanList=[18,19,20], mjd=1, style='p', ci=2)

signal([18,19,20], mjd=1, style='p', ci=2)

5.3.18 Plotting FFT

A Fast Fourrier Transform (FFT) of the signal can be plotted using the *fft* method:

```
read('spiral1')
op()                  # open an XWindow device
data.fft(range(10))   # plot FFT for the first 9 channels
```

5.4 MB-Fits to FITS file conversion

To convert an MB-Fits file to a FITS file in the MAMBO format you can use the command *mambo*. The current version does NOT use the data contained in the data object in Boa, but reads the input file (with default name = BoaB.currData.FileName) and converts it to the Mambo format. Therefore, this procedure is somewhat decoupled from Boa.

5.5 Scripts

As BoA provides the full functionality of python this allows the use of scripts. Scripts can be run with the *execfile()* function where the name of the file has to be given as string argument. The suffix of the file is arbitry.
Example: If you want to have a look at the time series of channels 10 to 30 succesively, create the following script with your preferred editor. Note that in python the contents of the for loop (like if blocks, method definitions, etc.) have to be indented.

```
# testBoa.py
indir('../Fits/')      # set the input directory
read('3543')         # read file 3543.fits
op()                   # open graphic display
for i in range(10,31): # start a for loop, the indentation in the
                       # following lines is mandatory
  sig([i])             # plot time series
  raw_input()          # wait for <Return>
```

To run the script type:

```
execfile('testBoa.py')
```

5.6 Abbreviations

[CV: I've made the relevant corrections to this table.]
[CV: There are also the following comments:
1) Note "unflagChannels" added
2) Note change to lower case "C" in "plotcorrel" command.
3) "restoreData" has many abbreviations.
4) Note the Description added for "signal"...check this is ok!
5) "statistics" caused a Segmentation fault and crashed boa! Seemed to be a one-off event though.]

Command	Abbreviations	Description
azel		plot elevation versus azimuth
azeloff	azelo	plot elevation offset versus azimuth offset
azimuth	azim ... az	plot azimuth versus LST
azimuthOffset	azimoff ... azo	plot azimuth offset versus LST
baseline	base	fit and subtract baseline
basesub		subtract baseline per subscan
chanmap		plot channel maps
channels	channel ... chan	select list of channels
clear	clea ... cle ... cl	clear the active plot window
close	clos ... clo	close one device
cormatrix	cmatrix	compute correlation matrix
correlate	cor	compute correlation relative to a reference channel
device	devic ... devi ... dev	select an open device
dumpData	dumpDat ... dumpD ... dump	save data object to a file
elevation	elev ... el	plot elevation versus LST
elevationOffset	eleoff ... elo	plot elevation offset versus LST
flag		flag data at more than n*rms
flagChannels	flagCh ... flagC ... fCh	flag a list of channels
flagLST		flag data by LST interval
flagLon		flag data by Az offset interval
indirls	ils	list input directory
indir	indi ... ind	set input directory
infile	infil ... infi ... inf	set input file
mambo		convert MB-Fits file to MAMBO format
mess		display a message
open	ope ... op	open a graphic device
outfile	outfil ... outfi ... outf	set output file
outdir	outdi ... outd	set output directory
plotcorrel	plotcor / plotCor	plot signal vs. reference channel
plotMean	plotmean	plot mean values vs. subscan numbers
plotMeanChan	plotmeanchan	plot mean values vs. chan. numbers
plotRms	plotrms	plot rms values vs. subscan numbers
plotRmsChan	plotrmschan	plot rms values vs. chan. numbers
read		read in a file
readRCPfile	readRCP ... rcp	read in an RCP file
resiz	resi	resize the plot, after resizing window with mouse
restoreData	restoreD ... restore ... restor	restore a previously stored BoA *.sav file
signal	signa ... sign ... sig	plot the time series of the data (flux density versus LST)
snf		compute and subtract skynoise
statistics	stat	prints the statistics
unflag		unflag data
unflagChannels	unflagCh ... unflagC ...ufCh	unflag a list of channels

Table 4: List of commands with abbreviations. Don't forget to add the round brackets () at the end of the commands.

6 Graphics: BoGLi

[CV: First a general comment...I've re-written/commented/updated everything that was in this Section 6 already. However, there are obviously a lot of things "missing" from this section, but whether they should be there or not depends on what the purpose of this Section is. For example, at the moment there are just a few examples of commands listed in 6.2 (See Table 5)...is this what is intended or do we in fact want to list here all the things Bogli can do?
]
[CV: These first two paragraphs (6.1 and 6.2) probably need re-writing a bit by someone with more experience of BoGli's development. My two comments are: 2nd sentence: "It's", and 3rd sentence: "self-consistent and may be used independantly within any python programme."
]

6.1 Introduction

The BoA Graphic Library (BoGLi) is an object-oriented software package for the graphical display of data. Its is written in Python and uses pgplot, the python binding to pgplot. The main parts (classes) of the software are self-consistent and may independently be used from any python programme. Nevertheless, BoGLi comes with features which especially customise its use for the display of astronomical data from multi-channel receivers. Its main goal is to provide a graphic tool tailored for the use with BoA for the display of data from LaBoCa, Simba and Mambo.

6.2 Commands

BoGLi has its own command handler. Nevertheless, anytime the BoA command handler encounters a graphic command this is automatically passed to the BoGLi command handler. Therefore, the user does not have to care about the separation between BoA and BoGLi commands. Table 5 gives an overview of some of the available commands.
[CV: Table 5 is intended to just show a few examples, right?...not all the available commands....
]
BoGLi provides a variety of attributes that may be changed by the user. The attribute name is then used as command followed by the desired value as argument (see Sect. 6.9 for details.)
[CV: I've updated this table. I've removed "default" and "limits" since I couldn't find them anywhere...are they obsolete?
]

Table 5: List of example BoGLi commands.

DeviceHandler.openDev	open a device
DeviceHandler.closeDev	close a device
Plot.clear	clear the active plot window
DeviceHandler.selectDev	select a device
DeviceHandler.resizeDev	resize the plot, after plot window resized using mouse
Plot.plot	make a single plot
MultiPlot.plot	plot multiple plots
Plot.draw	draw on an image
MultiPlot.draw	draw on plots of multiple channels

[CV: Any other commands to add? Are the two "draw" descriptions correct here???]

A detailed description of these commands is given in 6.3 below.
[CV: Should add here something like "The full list of available commands can be found in Chapter 8."]

6.3 Commands

[CV: In this section I've removed all the abbreviations since all the brackets were making it messy, and in any case the abbreviations are all clearly given in Table 4.]
[CV: Also, a lot of this section is simply repeating material already covered in Section 5, e.g. this is just a list of example commands such as clear and close...I think a clear distinction needs to be made between the purpose of Section 5 and Section 6.]

6.3.1 Opening a plot window

Usage: **DeviceHandler.openDev(***optional argument***)**

Optional argument: *pgplot device type*

[CV: First 2 sentences...someone check this is ok. I have rewritten 3rd/4th sentences. Also, where appropriate I have changed the itemised section below to what I think makes more sense.
]
Open a graphics device for pgplot output and make it the current device. If the device is opened successfully, it becomes the selected device to which graphics output is directed until another device is selected (see 6.3.4) or the device is closed (see 6.3.2). If no device argument is specified PGPLOT will open the default graphics device (an XWINDOW). Alternatively, the graphics device may be selected using any of the following as arguments:

(1) A complete device specification of the form 'device/type' or 'file/type', where /type is one of the allowed PGPLOT device types (installation-dependent, e.g. /xwindow) and 'device' or 'file' is the name of a graphics device or disk file appropriate for this type. The 'device' or 'file' may contain '/' characters; the final '/' delimits the 'type'. If necessary to avoid ambiguity, the 'device' part of the string may be enclosed in double quotation marks.

Example: **'plot.ps/ps', 'dir/plot.ps/ps', '"dir/plot.ps"/ps', 'user:[tjp.plots]plot.ps/PS'**

(2) A device specification of the form '/type', where /type is one of the allowed PGPLOT device types, e.g. /xwindow. PGPLOT supplies a default file or device name appropriate for this device type.

Example: **'/ps'** (PGPLOT interprets this as 'pgplot.ps/ps')

(3) A device specification with '/type' omitted; in this case the type is taken from the environment variable PGPLOT_TYPE, if defined (e.g., setenv PGPLOT_TYPE PS). Because of possible confusion with '/' in file-names, omitting the device type in this way is not recommended.

Example: **'plot.ps'** (if PGPLOT_TYPE is defined as 'ps', PGPLOT interprets this as 'plot.ps/ps')

(4) A blank string (' '); in this case, PGOPEN will use the value of environment variable PGPLOT_DEV as the device specification, or '/NULL' if the environment variable is undefined.

Example: **' '** (if PGPLOT_DEV is defined)

(5) A single question mark, with optional trailing spaces, i.e. ('?'). In this case, PGPLOT will prompt the user to supply the device specification, with a prompt string of the form 'Graphics device/type (? to see list, default XXX):' where 'XXX' is the default (value of environment variable PGPLOT_DEV).

Example: **'? '**

(6) A non-blank string in which the first character is a question mark (e.g. '?Device: '); in this case, PGPLOT will prompt the user to supply the device specification, using the supplied string as the prompt (without the leading question mark but including any trailing spaces).

Example: **'?Device specification for PGPLOT: '**

In cases (5) and (6), the device specification is read from the standard input. The user should respond to the prompt with a device specification of the form (1), (2), or (3). If the user types a question-mark in response to the prompt, a list of available device types is displayed and the prompt is re-issued. If the user supplies an invalid device specification, the prompt is re-issued. If the user responds with an end-of-file character, e.g., ctrl-D in UNIX, program execution is aborted; this avoids the possibility of an infinite prompting loop. A programmer should avoid use of PGPLOT-prompting if this behavior is not desirable.
The device type is case-insensitive (e.g., '/ps' and '/PS' are equivalent). The device or file name may be case-sensitive in some operating systems.

6.3.2 Closing a plot window

Usage: **DeviceHandler.closeDev(***optional argument***)**

Optional argument:

device number (integer)

'all'

'current'...'curre'...'cur'

Example:

DeviceHandler.closeDev(2): Close the device with identifier 2

DeviceHandler.closeDev('all'): close all devices

DeviceHandler.closeDev('current'): close current device (the default if no argument specified)

6.3.3 Clearing a plot window

Usage: **Plot.clear()**

Clear the output of the current device. To clear the output of a different device change to that device first (see 6.3.4).

6.3.4 Selecting a device

Usage: **DeviceHandler.selectDev(***argument***)**

Argument: *device number* (integer)

Select an open device for graphical output. The selected device has to be previously opened with **open** (see 6.3.1).

Example:

DeviceHandler.selectDev(2): Make device number 2 the current device for graphical output.

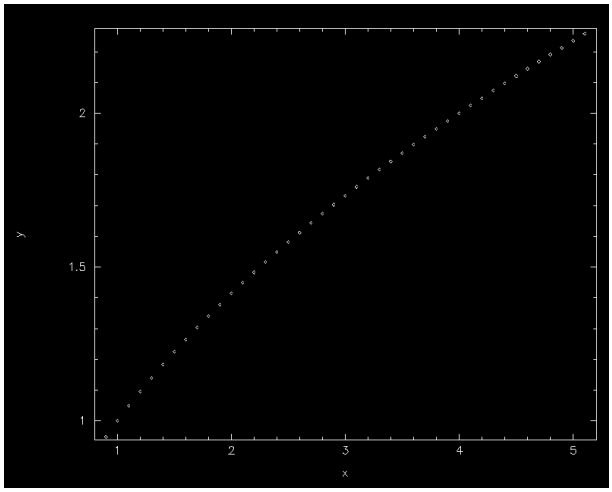

Figure 6.1: Example 1 of graphics produced using Plot.plot

6.3.5 Resizing a device

Usage: **DeviceHandler.resizeDev()**

Resize the plotting area after resizing of the graphics display window using the mouse. This is applicable to some interactive devices (e.g. /xwindow).

6.4 Plotting: single plots

[CV: Should maybe mention that all the following examples can be found in the file test.bogli.py, if this will be made available to everyone.
]
Usage: **Plot.plot(dataX**, *[dataY, limitsX, limitsY, labelX, labelY, caption, style, ci, width, overplot, aspect, logX, logY, nodata]* **)**

Optional arguments: these are described in Section 6.9.3. Note *dataY* is also optional – if no *dataY* is supplied the default is to plot dataX versus running number.
[CV: I guess the logical parameter will be described in a different section, but at the moment there is nothing in that section. So currently e.g. logX and logY are not described anywhere.]
[CV: Do we need to give a description here of each of these optional arguments or not (it's going to be listed in Chapter 8 so is described there)? Also, presentation

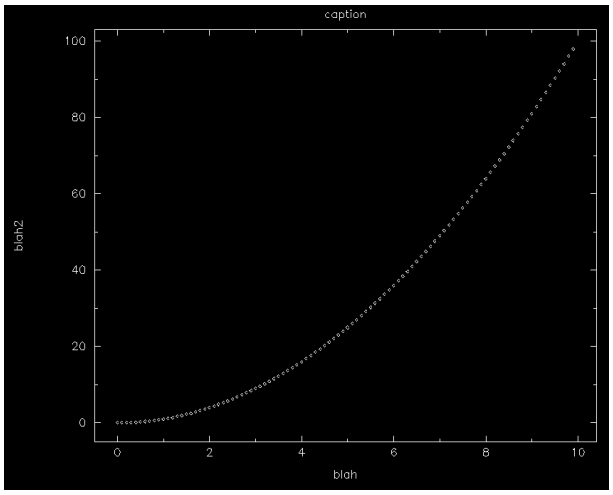

Figure 6.2: Example 2 of graphics produced using Plot.plot

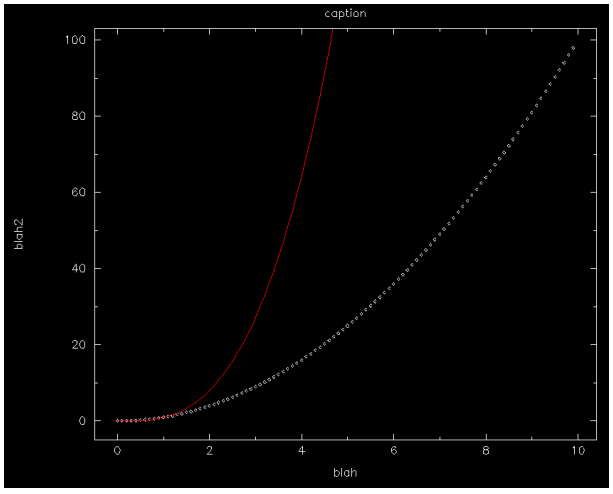

Figure 6.3: Example 3 of graphics produced using Plot.plot

style of optional arguments differs here from the lists given for some of the other commands, e.g. in Chapter 5. I'm not sure what's best at the moment - I'm trying to be consistent with the previous Chapter....we should change both chapters to make them consistent once all the info is in.
]
[**CV: Add a short description of what this command does.**
]
Example 1:

```
x = Numeric.array(range(100),Numeric.Float)/10

Plot.plot(x,Numeric.sqrt(x),limitsX=[1,5])
```

Note that Y limits are then computed according to this X range.

The graphic output produced in this case is shown in Figure 6.1.

Example 2:

```
Plot.plot(x,x*x,labelX='blah',labelY='blah2',caption='caption')
```

Note that plot clear the screen first, you need to use the new 'overplot' keyword (see below).

The graphic output produced in this case is shown in Figure 6.2.

Example 3:

```
Plot.plot(x,x*x*x,overplot=1,ci=2,style='l')
```

The graphic output produced in this case is shown in Figure 6.3.

6.5 Plotting: plot multiple channels

Usage: **MultiPlot.plot(chanList, dataX, dataY**, *[limitsX, limitsY,labelX,labelY, caption, style, ci, overplot, logX, logY, nan]* **)**

Arguments:

chanList: list of channels, of the form [1,2,3]

dataX: values to plot along X

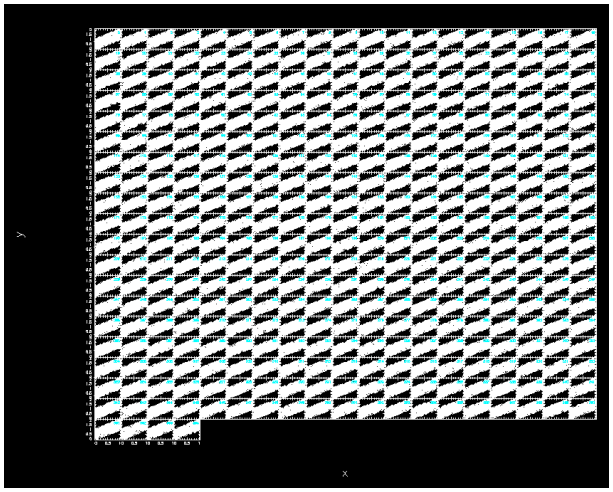

Figure 6.4: Example of graphics produced using MultiPlot.plot

dataY: values to plot along Y

Optional arguments: these are described in Section 6.9.3.
[CV: I guess the logical parameter will be described in a different section, but at the moment there is nothing in that section. So currently e.g. logX and logY are not described anywhere.
]
Example:

```
n_point = 365
chanlist=range(n_point)

x2 = RandomArray.random([n_point,n_point])
y2 = RandomArray.random([n_point,n_point])

MultiPlot.plot(chanlist,x2,y2+x2,style='p')
```

The graphic output produced in this case is shown in Figure 6.4.

6.6 Drawing on an image

Usage: **Plot.draw(map_array**, *[sizeX, sizeY, WCS, limitsX, limitsY, limitsZ, nan, labelX, labelY, caption, style, contrast, brightness, wedge, overplot, aspect, doContour, levels,*

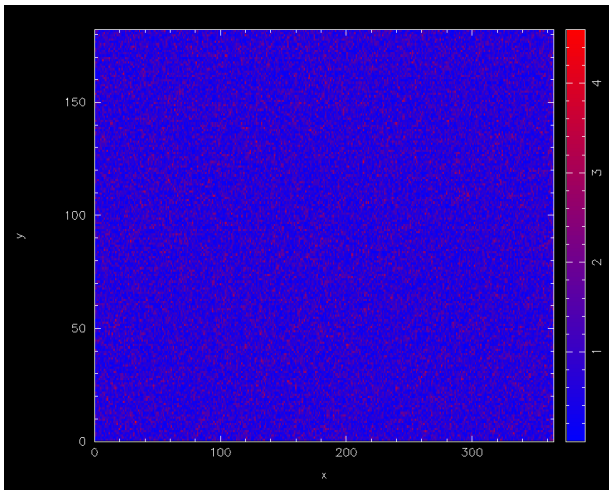

Figure 6.5: Example 1 of graphics produced using Plot.draw

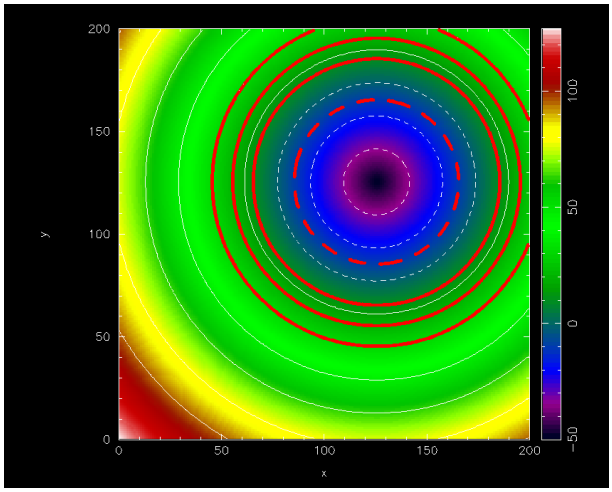

Figure 6.6: Example 2 of graphics produced using Plot.draw: drawing contours

labelContour])

Arguments:

map_array: map to display

Optional arguments: these are described in Section 6.9.3.

Example 1:

```
n_point = 365

mapping = Numeric.absolute(RandomArray.standard_normal([n_point,n_point/2]))

Plot.draw(mapping,style='b2r',wedge=1)

# You can also define 'physical' unit for your plot and still use
# limitsX/Y and aspect:

Plot.draw(mapping,sizeX=[-1,1],sizeY=[-2,2],limitsY=[-1,1],aspect=1, wedge=1)
```

The graphic output produced in this case is shown in Figure 6.5.

Example 2: Plotting contours
[CV: where can i find Plot.contour? I've moved this section to here (it was previously a section on its own) since it seems to me a part of "Plot.draw", since "doContour" is an optional argument of Plot.draw.
]

```
def dist(x,y):
    return (x-125)**2+(y-125)**2

image = Numeric.sqrt(Numeric.fromfunction(dist,(200,200)))-50

Plot.draw(image,wedge=1,aspect=1,style='rainbow')   # display an image
Plot.draw(image,doContour=1,overplot=1)             # overlay some contours
Plot.contour['color'] = 2                           # change the colour and
Plot.contour['linewidth'] = 10                      # linewidth attributes

Plot.draw(image,doContour=1,overplot=1,levels=[-10,10,20,30])  # plot some
      # more contours with the new attributes
```

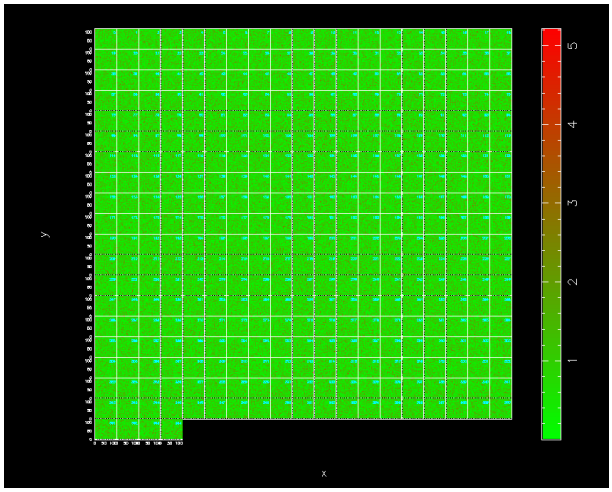

Figure 6.7: Example of graphics produced using MultiPlot.draw

The graphic output produced in this case is shown in Figure 6.6.

6.7 Drawing on plots of multiple channels

Usage: **MultiPlot.plot.draw(chanList,map_arrays**, *[sizeX, sizeY, WCS, limitsX, limitsY, limitsZ, nan, labelX, labelY, caption, style, contrast, brightness, wedge, overplot]* **)**

Arguments:

chanList: list of channels

map_arrays: lits of map to display

[CV: Re map_arrays above: this is what is written in MultiPlot.py, but I don't know what it is supposed to say..."lists of maps to display"????
]
Optional arguments: these are described in Section 6.9.3.

Example:

```
mapping_array = []
n_map = 365
```

```
for i in range(n_map):
    mapping_array.append(Numeric.absolute(RandomArray.standard_normal([120,120])))

MultiPlot.draw(range(n_map),mapping_array,wedge=1)
```

The graphic output produced in this case is shown in Figure 6.7.

6.8 Device handling

[CV: I've only changed the commands listed here so they are written as should be entered. But much of this section seems to me superfluous since almost all of it is described in the discussion of the individual commands in the previous section! Maybe this section should come before section 6.3, or maybe this should be just a short intro para on device handling and should then include the device handling commands now included in section 6.3. That just leaves the Boa commands such as azel and signal in what is now 6.3 so maybe a short intro para could be added to that as a separate section. Section 6.5 is then ok where it is.]

BoGLi is based on pgplot and as a consequence the number and type of available devices depends on the actual configuration. A list of supported devices is given at http://www.astro.caltech.edu/ tjp/pgplot/devices.html. During installation the device drivers have to be selected by editing the file *drivers.list*. As many device drivers are available on selected operating systems only, you should ensure that drivers you do not want are commented out (place ! in column 1) to avoid installation failures. A version of *drivers.list* used for a Linux PC can be found in Sect 3.4.4.

The command handler of BoGLi provides a set of commands to manage output devices. To open a new device use *open*. Pgplot will then prompt for the device type and suggest a default selection. To get a list of the available devices of your installation enter *'?'*. A newly opened device automatically becomes the current device.

To get a list of all opened devices and the current device identifier enter *device('?')*. Swapping between devices is also provided by the command *device*. Example: *device(3)* will switch to device number 3 if this is already opened. To close a device enter *close* followed by the device number to be erased. Example: *close(3)*. The number of the closed device becomes available again and will be used in turn by subsequent calls of *open*. To close the current device enter *close(current)* and the closure of all openend devices is achieved by *close(all)*.

The size of some interactive devices (e.g. /xwindow) may be changed using the mouse. To adjust the actual plotting area use *resize* afterwards.

6.9 Attributes

6.9.1 Basic concepts

[CV: I think most of this section is obsolete, so probably should be re-written by someone familiar with the development of Bogli. Same goes for section 6.5.2 I think.
]
[CV: I will check all the keywords and their descriptions are up-to-date.
]
BoGLi provides a variety of parameters which allow to change and customise the graphical output as regards primitives like colours, linestyles, character sizes as well as text output and general appearance. These parameters are class attributes of the software for which methods exist to easily set and change values with simple user commands. To get a list of all available parameters and their current settings type *show all*. To retrieve the current value of a single parameter use *parameter ?*, where *parameter* is the parameter name. Example: *chLabelX ?* will provide the character size of the x-label.
The usual way to set parameter values is to type the parameter name followed by the desired value. Example: *chLabelX 0.5* changes the character size of the x-label to a value of 0.5 (in units of 1/40 the height of the view surface). In case the parameter name is followed by no reasonable argument BoA provides an error message. For arguments outside the corresponding scope pgplot uses its intrinsic default values while BoGLi keeps the false value typed in. To use the default value of every parameter provided by BoGLi use *default*.
Depending on their arguments four kinds of parameters can be distinguished:

(1) float and integer parameters (see Sect. 6.9.2)

(2) logical parameters (see Sect. 6.9.5.1)

(3) string parameters (see Sect. 6.9.5.2)

(4) list parameters (see Sect. 6.9.5.3).

Float and integer parameters require floating point or integer numbers as arguments, respectively. If a float argument is given where an integer number is expected BoA provides an error message. Logical parameters require 0 or 1 as arguments and serve as switches. Example: *box 0* switches off the display of the box(es); to switch it on again use *box 1*. List parameters contain lists or arrays and have special commands to be changed (e.g. *plot channel*).

6.9.2 Float and integer parameters

The designation of almost each float and integer parameter follows a common scheme by assembling two keywords describing:

(1) the general attribute (general attribute keyword)

(2) the affected part of the graphical output (affected part keyword).

Where possible the keywords are designated following the argument names of the pglot subroutine descriptions. Example: *ciBox* defines the colour (ci = colour index) of the box where *ci* is the general attribute keyword and *Box* the object keyword.

6.9.3 General attribute keywords

The following list contains all general attribute keywords:

ch = character height. The character height scales the size of text, graph markers and tick-marks. It is given in multiples of about 1/40 the height of the view surface.

ci = colour index. The colour index is an integer in the range 0 to a device-dependent maximum. The default colour index is 1, usually white on a black background for monitor displays or black on a white background for printed hardcopies. Colour index 0 corresponds to the background colour. If the requested color index is not available on the selected device, colour index 1 will be used.

coord = character string coordinates. Sets the location of a character string along the specified edge of the corresponding viewport, as a fraction of the length of the edge.

disp = displacement. Scales the displacement of a character string or a wedge from the specified edge of the corresponding viewport, measured outwards from the viewport in units of the character height. Use a negative value to write inside the viewport, a positive value to write outside.

fjust = character string justification. Controls justification of a string parallel to the specified edge of the corresponding viewport.

fjust = 0.0: the left-hand end of the string will be placed at coord;

fjust = 0.5: the center of the string will be placed at coord;

fjust = 1.0: the right-hand end of the string will be placed at coord.

Other values between 0 and 1 give inter-mediate placing.

ls = line style. The line style is an integer in the range 1 to 5 with the following codes:

1: full line

2: dashed

3: dot-dash-dot-dash

4: dotted

5: dash-dot-dot-dot

The line style does not affect graph markers, text, or area fill.

lw = line width. The line width is specified in units of 1/200 (0.005) inch (about 0.13 mm) and must be an integer in the range 1-201. This parameter affects lines, graph markers and text.

symbol = graph marker symbol. The graph marker is an integer with the following codes:

-1, -2 : a single dot (diameter = line width)

-3..-31 : a regular polygon with ABS(symbol) edges (style set by fill style)

0..31 : standard marker symbols

32..127 : ASCII characters (in current font)

127..4000 : a Hershey symbol number

width = width (used for wedges only). Sets the total width of a wedge including annotation, in units of the character height.

xleft = x-coordinate of left hand. xleft, xright, ybot and ytop specify the size and position of the viewport in normalised device coordinates. Normalised device coordinates run from 0 to 1 in each dimension. The viewport is the rectangle on the view surface "through" which one views the graph.

xright = x-coordinate of right hand. (See **xleft**.)

ybot = y-coordinate of bottom edge. (See **xleft**.)

ytop = y-coordinate of top edge. (See **xleft**.)

6.9.4 Affected part keywords

The following list contains the keywords for those parts of the graphical output that can be customised by float and integer parameters:

Box

Caption

ChanNum = channel numbering

Cont = contours

Data = data lines

GloVP = global viewport

LabelX = x-label

LabelY = y-label

Point = data points

Wedge

6.9.5 Parameter list

Not any combination of general attribute and object keyword is effectively reasonable. Table 6 compiles the defined parameters that may actually be changed by the user sorted by object keyword.

Table 6: List of all float and integer parameters.

ciBox	colour index of box
lsBox	line style of box
lwBox	line width of box
chCaption	character height of caption
ciCaption	colour index of caption
coordCaption	string coordinates of caption
dispCaption	displacement of caption
fjustCaption	string justification of caption
ciChanNum	colour index of channel numbering
coordChanNum	string coordinates of channel numbering
dispChanNum	displacement of channel numbering
fjustChanNum	string justification of channel numbering
ciCont	colour index of contours
lsCont	line style of contours
lwCont	line width of contours
ciData	colour index of data lines
lsData	line style of data lines
lwData	line width of data lines
leftxGloVP	left x-coordinate of global viewport
rightxGloVP	right x-coordinate of global viewport
botyGloVP	bottom y-coordinate of global viewport
topyGloVP	top y-coordinate of global viewport
chLabelX	character height of x-label
ciLabelX	colour index of x-label
coordLabelX	string coordinates of x-label
dispLabelX	displacement of x-label
fjustLabelX	string justification of x-label
chLabelY	character height of y-label
ciLabelY	colour index of y-label
coordLabelY	string coordinates of y-label
dispLabelY	displacement of y-label
fjustLabelY	string justification of y-label
chPoint	character height of data points
ciPoint	colour index of data points
symbolPoint	marker symbol of data points
chWedge	character height of wedge
ciWedge	colour index of wedge
dispWedge	displacement of wedge
lsWedge	line style of wedge
lwWedge	line width of wedge
widthWedge	width of wedge

6.9.5.1 Logical parameters

6.9.5.2 String parameters

6.9.5.3 List parameters

7 Development

[CV: I've not looked at this Chapter because I'm not familiar with the development or whether or not things currently written here are up-to-date. However, if someone else does the necessary revisions I'd be happy to proof-read it for clarity etc.]

7.1 Basic programming rules

7.2 Adding classes

7.3 Adding methods

7.4 Adding Fortran90 code

FB040510

General

We are using Fortran 90/95 subroutines, wrapped to be called from python using the f2py package. This is because f90 code executes much faster than python scripts. There are some subtelties to pay attention to when wrapping fortran code, else you will add large overheads from the py-f90 interface, as arrays are copied and reindexed. For an introduction to F90/95 (only minor differences between the two), I recommend the compact and rather comprehensive (and free!) "Fortran 90 course notes"[6] by AC Marshall from the University of Liverpool. It contains all you probably need to know. I wrote a simple fortran method in BoA/fortran/BoaTest1.f90 to illustrate some basic features and give you a chance to test the wrapper without BoA. Look at its header for details. For an online F90/95 language reference[7] the best I found is at the NCSA resources page, describing IBM's XL Fortran for AIX 8.1 – which is close to the Intel compiler.

F90 in BoA

For BoA our general idea is to have one f90.so extension module, which includes all the f90 methods (called subroutines and functions in fortran). This is necessitated by that the f90.data module, which contains much of a scans data, is connected (through an "use data") to the other f90 program modules, and therefore they all need to be linked together.

The f90 methods may be split into different modules (classes) for convenience. We now have the first operational modules BoaF1.f90, BoaChannelAnalyser.f90, BoaBaseLine.f90, and the data module BoaData.f90. Each module may include any number of subroutines or functions. The data module BoaData.f90 is like a common block that contains all the data which does not

[6] http://math.nist.gov/~WMitchell/f90course/CourseNotes.pdf

[7] http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/IBMp690/IBM/usr/share/man/info/en_US/xlf/html/lr02.HTM#CONTENT

change during data reduction. All data which does change is passed to the fortran subroutines as call arguments.
The BoaData.f90 (f90.data from python) module is filled in BoaDataEntity.FillF90. It must be refilled if you change data object, else the fortran methods will work on a different scan. This re-filling must be implemented still. Currently the f90.data is only filled upon read of a new data file.
The CVS directory BoA/fortran contains the fortran source code. You will need to wrap/compile the BoA modules on your local system (see below), since it links to local libraries that have no standard address. This will create the extension module f90.so which you import to BoA. From the CVS directory BoA start BoA, then

```
>>> from fortran import f90
```

This is how to import any module from a subdirectory, which for this needs to include an empty file __init__.py
The python script fortran/ftest.py contains a series of calls to the fortran subroutines. To run it:

```
>>> read()  # read in some scan
>>> op()    # open plot device
    [enter]
>>> execfile('ftest.py')  # start the script
```

which is followed with lots of output. To illustrate the use of new python methods that use fortran, you find BoA/TestFB.py, which you run like ftest.py. It goes through a number of data reduction steps and plots the data.

Wrapping F90 code with f2py

To wrap the f90 modules to produce f90.so:

```
ifc -c -w svd.f90
f2py -c -m f90 BoaData.f90 BoaF1.f90 BoaChannelAnalyser.f90 BoaBaseLine.f90 svd.o
     or on some installations alternatively:
f2py -c --fcompiler=intel -m f90 BoaData.f90 BoaF1.f90 BoaChannelAnalyser.f90 BoaBaseL
```

The first command recompiles the svd.o. On the f2py line there are some diagnostic options you may add if you debug your code:

```
-DF2PY_REPORT_ATEXIT : gives time statistics upon exit from python.
-DF2PY_REPORT_ON_ARRAY_COPY=1000 : reports when the f2py interface copies an array.
-DNUMARRAY : must be used for numarray support. Default is Numeric.
```

If the wrapping fails, one of the following may be wrong:
1. You have not initiated the ifc compiler properly. In your shell initialization file (e.g. .cshrc for tcsh) you need

```
if (-e /opt/intel/compiler60/ia32/bin/ifcvars.sh) then
  source /opt/intel/compiler60/ia32/bin/ifcvars.csh
endif
```

or something equivalent.
2. Your python path does not include the intel fortran compiler:

```
setenv PYTHONPATH ".:/opt/intel/compiler60/ia32/lib/:
                    /usr/local/lib/python2.3:
                    /usr/local/lib/python2.3/site-packages:
                    /home/bertoldi/bin:
                    /opt:
                    /usr/lib"
```

3. You use an old version of f2py.

```
<fortran> f2py -version
2.39.235_1644
```

Once you have successfully imported f90 in BoA, you can inquire about the use of a given method by typing

```
print f90.f1.NAME.__doc__
```

Fortran attributes are called f90.data.name_of_attribute. To inquire which ones are available:

```
boa> print f90.data.__doc__
el - 'f'-array(218)
track_el - 'f'-array(218)
ffcf_gain - 'f'-array(120)
subscan_time - 'f'-array(4)
az_p - 'f'-array(109,3)
lst - 'f'-array(218)
lon_p - 'f'-array(109,3)
track_az - 'f'-array(218)
lat - 'f'-array(218)
az - 'f'-array(218)
lat_p - 'f'-array(109,3)
```

```
lst_p - 'f'-array(109,3)
array_gain - 'f'-array(120)
lon - 'f'-array(218)
ffcf_cn - 'f'-array(120)
ut_p - 'f'-array(109,3)
nodding_sta - 'i'-array(218)
subscan_index - 'i'-array(4)
subscan_num - 'i'-array(4)
weights - 'f'-array(0), not allocated
el_p - 'f'-array(109,3)
ut - 'f'-array(218)
wobbler_pos - 'f'-array(218)
```

They are filled in in BoaBusiness.py: BoaB.FillF90

Use f90 methods in BoA

To call a fortran method, here an example:

```
compressed_array,nmax = f90.f1.compress(array,flag_array,0)
```

Two objects are returned as a tuple, an array and an integer. They both are not in the call argument list, they are hidden to python, but are listed in the f90 code call argument list – have a look at the source code.

Limitations

This particular example illustrates one of the limitations of wrapping f90 code: you cannot return an array with a length that is determined upon execution. The wrapper needs to specify the size of an array somehow. It does not have to be fixed, but specified through the size of an input attribute at least. In this example we try to return an array that is a compression of the input array, determined by the condition that the corresponding flag is 0. The trick to still do this here is to return a comressed_array with the same size as array, plus an integer telling the size of the compressed array, so that the final answer is compressed_array[0:nmax].

Fortran vs. C-contiguous

If a Numeric array is proper-contiguous and has a proper type then it is directly passed to the wrapped Fortran function. Otherwise, an element-wise copy of an input array is made and the copy, being proper-contiguous and with proper type, is used as an array argument. There are two types of proper-contiguous Numeric arrays: Fortran-contiguous arrays when data is stored column-wise, i.e. indexing of data as stored in memory starts from the lowest dimension; C-contiguous when data is stored row-wise, i.e. indexing of data as stored in memory starts from the highest dimension. For one-dimensional arrays these notions coincide. To transform input arrays to column major storage order before passing them to Fortran routines, one may use the function as_column_major_storage(¡array¿) that is provided by all

F2PY generated extension modules, such as the BoA f90. If you call a fortran method repeatedly with the same input array, you should convert the array first to avoid conversion by the wrapper interface on each call – which could dominate the execution time here. If you add the option -DF2PY_REPORT_ON_ARRAY_COPY=1000 when wrapping, you will be informed on each copy that the wrapper interface performs. The option -DF2PY_REPORT_ATEXIT gives an execution time summary upon exit that splits up the time used in fortran and in the interface. If the interface time is large or comparable to the fortran execution time, your code is not efficient because it copies arrays too often. Look at examples in BoaBaseLine.py, e.g.:

```
Data = f90.as_column_major_storage(self.Data.Data_Red_p)
Flag = f90.as_column_major_storage(self.Data.Data_Flag_p)
...
for i_ch in ch_range: # loop over channels and phases
    for i_ph in ph_range:
      Data = f90.baseline.addpoly(Data,Poly,Mean,Rms,i_ph,i_ch)
```

The input arrays are copied once into fortran-contiguous arrays before the loop, so in the loop there is no overhead from copying. Note also the general scheme of calling a fortran method here: Data is in- and output argument.

7.5 Interfacing

7.6 ScientificPython-2.4.5

ScientificPython is a collection of Python modules that are useful for scientific computing. Almost all modules make extensive use of Numerical Python (NumPy,Numeric), which must be installed prior to Scientific Python. Scientific constist of about one dozen modules, which contain methods written in Python that may come handy, but may be slow. The following lists a number of them.
stat() statistics() command calculates the statistics for all the channels in the range. Using plotmean() plotrms() we can plot mean and RMS values of each channels. The examples are as shows below:
You need to import Numeric for Scientific. You can access the methods by importing the class or all methods:

```
>>> from Numeric import *
>>> import Scientific.Statistics
>>> Scientific.Statistics.median([1,2,3,5,6])
3.0
```

or alternatively

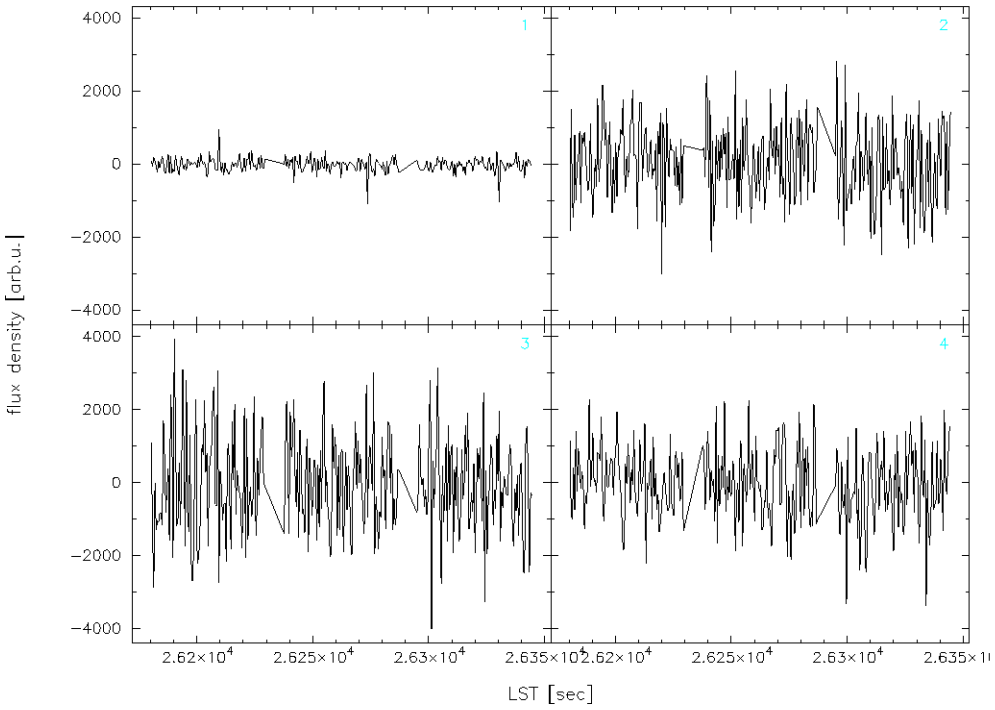

Figure 7.1: Plotting the Signal for channels in the range.

```
>>> from Scientific.Statistics import *
>>> median([1,2,3,5,6])
3.0
```

Available method in class Scientific.Statistics:

```
moment(data, order, about=None, theoretical=1)
mean(data)
weightedMean(data, sigma)
variance(data)
standardDeviation(data)
median(data)
mode(data)
normalizedMoment(data, order)
```

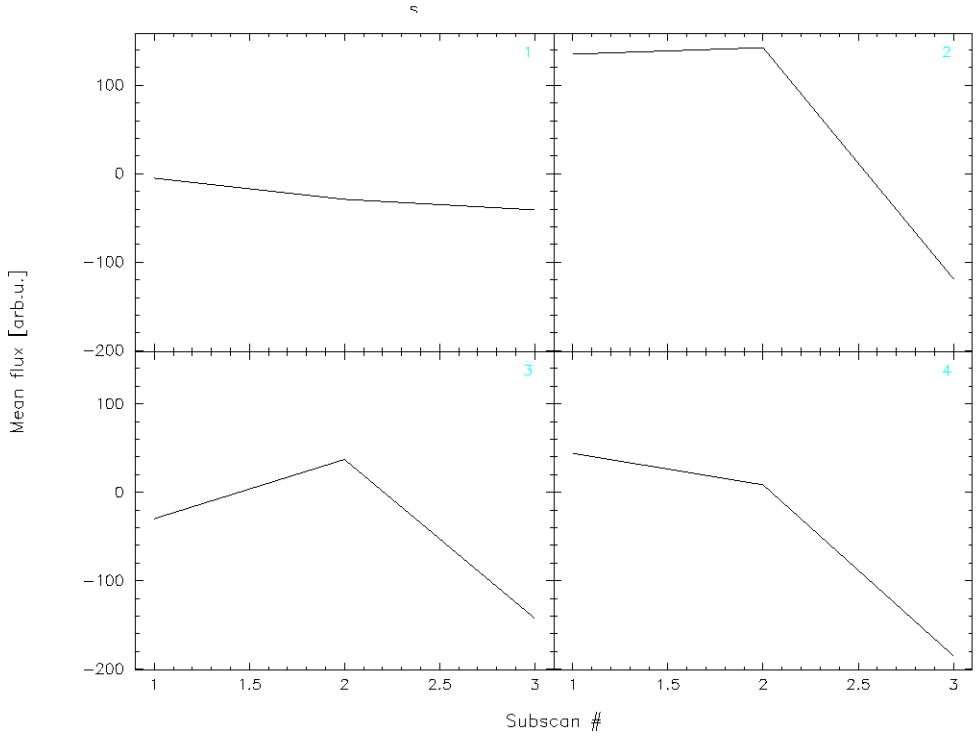

Figure 7.2: Plotting the Mean values of signal.

```
skewness(data)
kurtosis(data)
correlation(data1, data2)
```

There are also two classes for histograms:

```
Histogram
WeightedHistogram(Histogram)
```

The following explains only those Scientific methods which are useful for Boa. Consult the scripts or the (very sparse) documentation for more info.

7.6.1 Scientific.Statistics.median

Description: Computes the median of a 1-d array.

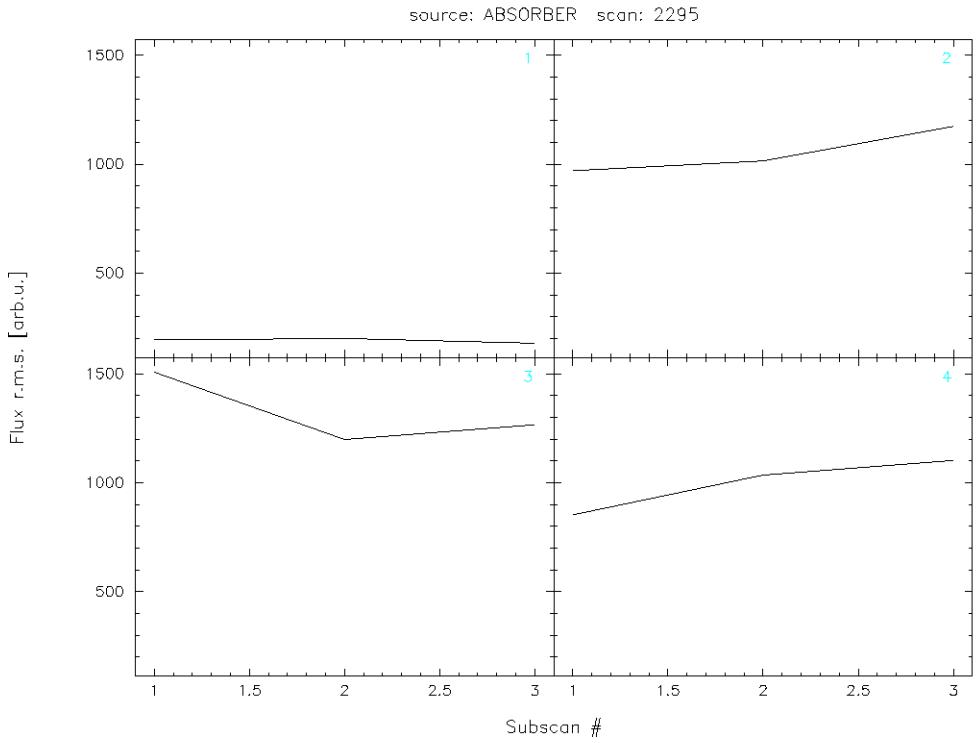

Figure 7.3: Plotting the RMS values of signal.

Example:

```
>>> median([1,2,3,5,6])
3.0
```

7.6.2 Scientific.Statistics.mean

Description: Returns the mean (average value) of a 1-d array.
Example:

```
>>> mean([1,2,3,5,6])
3.3999999999999999
```

7.6.3 Scientific.Statistics.correlation

Description: Computes the correlation coefficient between two 1-dim arrays a and b according to

$$c_{ab} = \frac{\langle (a - \bar{a})(b - \bar{b}) \rangle}{\langle (a - \bar{a})^2 \rangle^{1/2} \langle (b - \bar{b})^2 \rangle^{1/2}} \tag{7.1}$$

Example:

```
>>> correlation([1,2,3,4,5],[1,2,3,4,5])
1.0
>>> correlation([1,2,3,4,5],[1,2,3,5,5])
0.96476382123773219
>>> correlation([1,2,3,4,5],[5,4,3,2,1])
-1.0
```

7.6.4 Scientific.Functions.LeastSquares

Description: General non-linear least-squares fit using the Levenberg-Marquardt algorithm and automatic derivatives. The parameter model specifies the function to be fitted. It will be called with two parameters: the first is a tuple containing all fit parameters, and the second is the first element of a data point (see below). The return value must be a number. Since automatic differentiation is used to obtain the derivatives with respect to the parameters, the function may only use the mathematical functions known to the module FirstDerivatives. The parameter parameter is a tuple of initial values for the fit parameters. The parameter data is a list of data points to which the model is to be fitted. Each data point is a tuple of length two or three. Its first element specifies the independent variables of the model. It is passed to the model function as its first parameter, but not used in any other way. The second element of each data point tuple is the number that the return value of the model function is supposed to match as well as possible. The third element (which defaults to 1.) is the statistical variance of the data point, i.e. the inverse of its statistical weight in the fitting procedure. The function returns a list containing the optimal parameter values and the chi-squared value describing the quality of the fit.

Example:

```
>>> from Numeric import exp
>>> def f(param, t):
...     return param[0]*exp(-param[1]/t)
...
>>> data = [(100, 4.999e-8),(200, 5.307e+2),
  (300, 1.289e+6),(400, 6.559e+7)]
```

```
>>> print leastSquaresFit(f, (1e13,4700), data)
([8641551709749.7666, 4715.4677901570467], 1080.2526437958597)
```

8 Description of all classes and methods

In the following all modules with their associated classes and methods are listed and described.

8.1 chanana

chanana < BoaChanAna.f90

```
! NAM: chanana (module f90) in file BoaChanAna.f90
!
! DES: module collecting f90 subroutines:
!        flag_pc
!        flag
!        unflag
!        mean_rms
!        mean_rms_s
!        statistics_by_subscan
!        flagtime
!        flaglon
!
! USE: f90.chanana.flag(...)  etc. See examples in ftest.py
!      For compilation instruction see BoaData.f90
!
! HIS: FB040426  assemble flag,mean_rms,polyfit in one module
!      FB040501  add and revise subroutines. data module reduced to constant attributes
!      FB040508  major revisions to some routines
!      FB040509  add  statistics_by_subscan
!      FB040515  split BoaF1.f90 into several files corresponding to classes
!      FB040531  move cmatrix to BoaSNF.f90
!      FB040602  add unflag
!      FB040603  add flagtime
!      FB040608  add flaglon
!      FB040611  statistics_by_subscan: use rank-2 sub_index
```

History:

```
! NAM: chanana (module f90) in file BoaChanAna.f90
!
! DES: module collecting f90 subroutines:
!        flag_pc
!        flag
!        unflag
!        mean_rms
```

```
!        mean_rms_s
!        statistics_by_subscan
!        flagtime
!        flaglon
!
! USE: f90.chanana.flag(...)  etc. See examples in ftest.py
!      For compilation instruction see BoaData.f90
!
! HIS: FB040426  assemble flag,mean_rms,polyfit in one module
!      FB040501  add and revise subroutines. data module reduced to constant attributes
!      FB040508  major revisions to some routines
!      FB040509  add  statistics_by_subscan
!      FB040515  split BoaF1.f90 into several files corresponding to classes
!      FB040531  move cmatrix to BoaSNF.f90
!      FB040602  add unflag
!      FB040603  add flagtime
!      FB040608  add flaglon
!      FB040611  statistics_by_subscan: use rank-2 sub_index
```

8.1.1 flag_pc(xflag,xdata,n_phase,n_channel,low,high,ii,jj,kk)

flag_pc < chanana < BoaChanAna.f90

```
! NAM: flag_pc  (sub)
! DES: set flag for data point of given channel if data <low or >high
! INP:
!      n_ph    = phase: 0=differences,1=one,2=two
!      n_ch    = channel number for which data to be flagged.
!                NOTE: python N corresponds to fortran N+1.
!                      channel 1 is index 0 in python arrays.
!      low     = lower limit
!      high    = upper limit
! OUT:
! HIS: FB040303  change to _p
!      FB040426  make indexing consistent with python
```

8.1.2 flag(xflag,xdata,low,high,mask_count,ii)

flag < chanana < BoaChanAna.f90

```
! NAM: flag  (sub)
! DES: set flag if data <low or >high
! INP:
!      low     = lower limit
!      high    = upper limit
! OUT:
! HIS: FB040501  adapted from flag_pc
```

8.1.3 unflag(xflag,xdata,flag,mask_count,ii)

unflag < chanana < BoaChanAna.f90

```
! NAM: flag  (sub)
! DES: set flag if data <low or >high
! INP:
!       xdata : data array
!       xflag : flag array
!       flag  : flag value to unflag
! OUT:  xflag : flag array
!       mask_count : number of elements changed
!
! HIS: FB040602  adapted from flag
```

8.1.4 flagtime(xflag,low,high,flag,mask_count,ii)

flagtime < chanana < BoaChanAna.f90

```
! NAM: flagtime  (sub)
! DES: set flag if LST>low and <high
! INP:
!      low     = lower limit
```

```
!      high    = upper limit
!      xflag   = rank-1 flag array
! OUT:
!      xflag   = rank-1 flag array
!      mask_count = number of flags changed
!
! HIS: FB040603  adapted from flag
```

8.1.5 flaglon(xflag,low,high,flag,mask_count,ii)

flaglon < chanana < BoaChanAna.f90

```
! NAM: flagaz  (sub)
! DES: set flag where lon>low and <high
! INP:
!      low     = lower limit
!      high    = upper limit
!      xflag   = rank-1 flag array
!      lon_p   taken from data module
! OUT:
!      xflag   = rank-1 flag array
!      mask_count = number of flags changed
!
! HIS: FB040608  adapted from flagtime
```

8.2 BoaConfig (module)

8.2.1 setInDir(inputDirectory=",r=0):

setInDir < Class name missing < BoaConfig.py

```
    DES: set the input directory
    INP: (string) inputDirectory = name of input directory
```

History:

```
NAM: BoaConfig (module)
DES: Set the path/filename for i/o operation
```

8.2.2 setOutDir(outputDirectory=",r=0):

setOutDir < Class name missing < BoaConfig.py

```
    DES: set the output directory
    INP: (string) outputDirectory = name of output directory
```

8.2.3 listInDir():

listInDir < Class name missing < BoaConfig.py

```
    DES: list the input directory
```

8.2.4 setInFile(inputFile='?',r=0):

setInFile < Class name missing < BoaConfig.py

```
    DES: set the input file name
    INP: (string) inputFile = name of input file
    OUT: (string) inputFile
```

8.2.5 setOutFile(outputFile='?',r=0):

setOutFile < Class name missing < BoaConfig.py

```
    DES: set the output file name
    INP: (string) outputFile = name of output file
    OUT: (string) outputFile
```

8.3 data

data < BoaData.f90

```
! NAM: data (module f90)
! HIS: FB031213  created
!      FB040209  add channel_rms/mean
!      FB040303  add _p attributes for phases/phase difference
!      FB040428  add subscan_*
!      FB040515  add message_unit
!      FB040610  add second index to xsubscan_index
!
! DES: f90 module for data interface
! USE:
!      Wrap with f2py:
! ifc  -c -w svd.f90         compile module -> svd.d  svd.o
! f2py -c --fcompiler=intel -m f90 BoaData.f90 BoaF1.f90 BoaFortranBaseLine.f90 BoaChanA
!
!      Options:  -DF2PY_REPORT_ATEXIT : gives time stats upon exit from python
!                -DF2PY_REPORT_ON_ARRAY_COPY=1000 : reports when f2py interface copies a
!                -DNUMARRAY : must be used for numarray support
```

History:

```
! NAM: data (module f90)
! HIS: FB031213  created
!      FB040209  add channel_rms/mean
!      FB040303  add _p attributes for phases/phase difference
!      FB040428  add subscan_*
!      FB040515  add message_unit
!      FB040610  add second index to xsubscan_index
!
! DES: f90 module for data interface
! USE:
!      Wrap with f2py:
! ifc  -c -w svd.f90         compile module -> svd.d  svd.o
! f2py -c --fcompiler=intel -m f90 BoaData.f90 BoaF1.f90 BoaFortranBaseLine.f90 BoaChanA
!
!      Options:  -DF2PY_REPORT_ATEXIT : gives time stats upon exit from python
```

```
!                -DF2PY_REPORT_ON_ARRAY_COPY=1000 : reports when f2py interface copies a
!                -DNUMARRAY : must be used for numarray support
```

8.4 DataAna(BoaDataEntity.DataEntity):

DataAna(BoaDataEntity.DataEntity): < BoaDataAnalyser.py

```
NAM: DataAna (class)
DES: An object of this class is responsible for the flagging of individual
     channels, i.e. it sets the values in the Channel_Flag array of the
     corresponding DataEntity object. It provides methods to derive the rms
     of each channel and to automatically search for bad or noisy channels.
     Channels might be flagged according to a given input file. This object
     provides methods to derive the correlation matrix.

     Uses:
           f90.chanana.mean_rms
           f90.chanana.mean_rms_s
           f90.chanana.flag
```

History:

```
NAM: BoaDataAnalyser.py (file)
DES: contains BoA channel analyser class
```

8.4.1 __init__(self):

__init__ < DataAna(BoaDataEntity.DataEntity): < BoaDataAnalyser.py

```
   DES: initialise an instance
```

8.4.2 flagChanFromList(self,list=[],flag=1):

flagChanFromList < DataAna(BoaDataEntity.DataEntity): < BoaDataAnalyser.py

```
   DES: assign flags to a list of channels
        To unflag a channel simply flag with flag=0
   INP: (integer) list=list of channels to be flagged
        (integer) flag=flag value
```

8.4.3 unflag(self,channel=-1,phase=-1,flag=1):

unflag < DataAna(BoaDataEntity.DataEntity): < BoaDataAnalyser.py

```
   NAM: unflag (method)
   DES: Unflag data, i.e. reset to 0.
Might want to change to allow for list of channels
   INP:
        (i) channel  =  -1 : all channels; 0=first channel
        (i) phase    =  -1 : all; else: 0,1,2
        (i) flag     =   1 : unflag only this value
```

8.4.4 flag(self,channel=-1,phase=-1,below=2,above=2):

flag < DataAna(BoaDataEntity.DataEntity): < BoaDataAnalyser.py

```
   NAM: flag (method)
   DES: Flag yet unflagged data below 'below'*rms and above 'above'*rms.
Might want to change to allow for list of channels
    INP:
        (i) channel  =  -1: all channels; 0=first channel
        (i) phase    =  -1: all; else: 0,1,2
        (f) below    = flag data with value < 'below'*rms
(f) above    = flag data with value > 'above'*rms
```

8.4.5 flagLST(self,channel=-1,phase=-1,below='?',above='?',flag=1):

flagLST < DataAna(BoaDataEntity.DataEntity): < BoaDataAnalyser.py

```
   NAM: flagLST (method)
   DES: Flag data in time interval
   INP:
        (i) channel  =  -1: all channels; 0=first channel
        (i) phase    =  -1: all; else: 0,1,2
```

```
         (f) below    = flag data
 (f) above    = flag data
```

8.4.6 flagLon(self,channel=-1,phase=-1,below='?',above='?',flag=1):

flagLon < DataAna(BoaDataEntity.DataEntity): < BoaDataAnalyser.py

```
    NAM: flagLon (method)
    DES: Flag data in azimuth offset (longitude) interval
    INP:
         (i) channel  =  -1: all channels; 0=first channel
         (i) phase    =  -1: all; else: 0,1,2
         (f) below    = flag data below and above
 (f) above    = flag data ...
```

8.4.7 statistics(self):

statistics < DataAna(BoaDataEntity.DataEntity): < BoaDataAnalyser.py

```
    NAM: statistics (method)
    DES: computes mean,median,rms for all scans and subscans
```

8.4.8 corMatrix(self):

corMatrix < DataAna(BoaDataEntity.DataEntity): < BoaDataAnalyser.py

```
    DES: compute correlation matrix
    INP:
```

8.4.9 correlate(self,phase=-1,channel=1,plot=1):

correlate < DataAna(BoaDataEntity.DataEntity): < BoaDataAnalyser.py

```
   NAM: correlate (method)
   DES: compute correlation factor relative to given reference channel
        for given phase
   INP:
(i) channel  =  reference channel against which correlated
        (i) phase    =  0,1,2
        (l) plot     =  1 or 0; overwrite Data_Red with fit?
   OUT: (f) data     =  overwrites Data_Red_p with fit
            FFCF_CN  =  rank-1 flat field with respect to channel
```

8.4.10 snf(self,phase=-1,channel=1,subtract=1,wa=0.95,wb=1.0):

snf < DataAna(BoaDataEntity.DataEntity): < BoaDataAnalyser.py

```
   NAM: snf (method)
   DES: compute correlated noise and subtract it from data
        currently for only one specified channel and phase
   INP:
(i) channel  =  reference channel for which CN is computed
        (i) phase    =  0,1,2
        (l) subtract =  1 or 0; subtract CN or not
        (f) wa       =  parameter for weights, usually = 0.90-0.98
        (f) wb       =  parameter for weights, usually = 1
```

8.4.11 basePolySubscan(self,channel=-1,phase=-1,order=1,subtract=1,plot=0):

basePolySubscan < DataAna(BoaDataEntity.DataEntity): < BoaDataAnalyser.py

```
   NAM: basePolySubscan (method)
   DES: polynomial baseline method, treat each subscan.
   INP:
```

```
     (i) channel  =  -1: all channels; 0=first channel
     (i) phase    =  -1: all; else: 0,1,2
     (i) order    =  polynomial order, >0
     (l) subtract =  1 or 0; subtract fit from data?
     (l) plot     =  1 or 0; overwrite Data_Red with fit?
OUT: (f) data     =  overwrites data array with fit or data-fit
     (f) poly     =  polynomial coefficients are printed but not
                     returned, needs to be implemented
```

8.4.12 basePoly(self,channel=-1,phase=-1,order=1,subtract=1,plot=0):

basePoly < DataAna(BoaDataEntity.DataEntity): < BoaDataAnalyser.py

```
NAM: basePoly (method)
DES: polynomial baseline method for single scans
INP:
     (i) channel  =  -1: all channels; 0=first channel
     (i) phase    =  -1: all; else: 0,1,2
     (i) order    =  polynomial order, >0
     (l) subtract =  1 or 0; subtract fit from data?
     (l) plot     =  1 or 0; overwrite Data_Red with fit?
OUT: (f) data     =  overwrites data array with fit or data-fit
     (f) poly     =  polynomial coefficients are printed but not
                     returned, needs to be implemented
```

8.4.13 addPoly(self,channel=0,phase=1,poly=[10.,0.,10.,10.]):

addPoly < DataAna(BoaDataEntity.DataEntity): < BoaDataAnalyser.py

```
NAM: addPoly (method)
DES: add polynomial baseline
INP:
     (int)      channel  =  channel. for -1 all channels. first channel=0
     (int)      phase    =  phase to be used. For -1 all. else: 0,1,2
     (int)      poly     =  polynomial coefficients, relative to
                            current mean and rms
```

8.5 DataEntity:

DataEntity: < BoaDataEntity.py

```
NAM: DataEntity (class)
DES: Objects of this class store the data and associated
     parameters of a scan, which can contain several observations
     (or subscans).
     They also contain additional arrays in which the current
     results of the data reduction are stored.
     This class also provides the interface between the MB-FITS
     files and BoA, by the means of the fillFromMBFits() method.
```

History:

```
NAM: BoaDataEntity.py (file)
DES: contains the BoA data entity class
```

8.5.1 __init__(self):

__init__ < DataEntity: < BoaDataEntity.py

```
DES: Instanciation of a new DataEntity object.
     All attributes are defined and set to default values.
```

8.5.2 reset(self):

reset < DataEntity: < BoaDataEntity.py

Attention: DES is missing! Please check the source code.
TXT:

```
Reset all attributes - useful before reading a new file
```

8.5.3 __str__(self):

__str__ < DataEntity: < BoaDataEntity.py

Attention: DES is missing! Please check the source code.
TXT:

```
Defines a string which is shown when the print instruction
is used.
It contains the sizes and typecodes of all attributes.
```

8.5.4 read(self,inFile='',readObs=[],update=0):

read < DataEntity: < BoaDataEntity.py

```
DES: fill a data entity object
INP: readObs (int list) observation numbers to read (default: all)
     update (logical) if true, do not reset previous entity object
```

8.5.5 fillFromMBFits(self,obsEntity,update=0):

fillFromMBFits < DataEntity: < BoaDataEntity.py

```
NAM: fillFromMBFits()
DES: fill a DataEntity object using the content of ObsEntity
     objects.

Calling sequence: DataEntity.fillFromMBFits(obsEntity)
INP: obsEntity: *LIST* of objects of the Entities.ObsEntity class
     update (logical) if true, do not reset previous entity object
OUT: None
```

8.5.6 FillF90(self):

FillF90 < DataEntity: < BoaDataEntity.py

```
NAM: FillF90  (method)
DES: Fill the f90 data module
INP: self = DataEntity object
```

8.5.7 dumpData(self,fileName='BoaData.sav'):

dumpData < DataEntity: < BoaDataEntity.py

```
DES: save the current DataEntity object to a file
INP: (string) fileName: name of the output file
     optional - default value = 'BoaData.sav'
```

8.5.8 restoreData(self,fileName='BoaData.sav'):

restoreData < DataEntity: < BoaDataEntity.py

```
DES: restore a DataEntity object previously saved in a file, and
     set it as the currData attribute of BoaB
INP: (string) fileName: name of the input file
     optional - default value = 'BoaData.sav'
```

8.5.9 saveMambo(self,inName='',outName=''):

saveMambo < DataEntity: < BoaDataEntity.py

```
DES: convert an MB-Fits file to the MAMBO FITS format, readable
     by MOPSIC
INP: (str) inName: name of the MB-Fits file (optional)
     (str) outName: name of the MAMBO output file (optional)
```

8.5.10 readRCPfile(self,rcpFile):

readRCPfile < DataEntity: < BoaDataEntity.py

```
NAM: readRCPfile (method)
DES: update Receiver Channel Parameters (attributes Array_Geo,
     Array_Gain and Channel_Sep) from the content of a file
INP: self = DataEntity object
     (string) rcpFile: complete name of file to read in
```

8.5.11 writeRCPfile(self,rcpFile='rcpBoa.rcp'):

writeRCPfile < DataEntity: < BoaDataEntity.py

```
NAM: writeRCPfile (method)
DES: store current Receiver Channel Parameters (Array_Geo,
     Array_Gain) to a file with format like mopsi
INP: (string) rcpFile: complete name of output file
```

8.5.12 initPhases(self):

initPhases < DataEntity: < BoaDataEntity.py

```
NAM: initPhases (method)
DES: Compute the phase differences from the raw data.
     New attributes are defined here, with names similar to
     the initial ones, with _p appended. They are 2D or 3D
     arrays, where the 1st dim. corresponds to integrations,
     the 2nd dim. means:

     0 = phase diff,
     1 = phase1,
     2 = phase2

     and the 3rd dim. corresponds to pixels, when relevant.
INP: self = DataEntity object
```

8.5.13 phaseDiff(self):

phaseDiff < DataEntity: < BoaDataEntity.py

```
NAM: phaseDiff (method)
DES: Recompute the phase differences on the Data_Red and
     Data_Noi attributes.
INP: self = DataEntity object
```

8.5.14 copyPhases(self):

copyPhases < DataEntity: < BoaDataEntity.py

```
NAM: copyPhases (method)
DES: when only one phase, copy data to phase2 and phase diff
INP: self = DataEntity object
```

8.5.15 existData(self):

existData < DataEntity: < BoaDataEntity.py

```
DES: check if the DataEntity object has been filled with data
INP: none
OUT: (int) result: 0 if no data, 1 otherwise
```

8.5.16 getTotalChanList(self):

getTotalChanList < DataEntity: < BoaDataEntity.py

```
DES: generate and get the complete list of available channels
INP: -
OUT: none
```

8.5.17 setCurrChanList(self,chanList='?'):

setCurrChanList < DataEntity: < BoaDataEntity.py

```
DES: set list of channels to be treated
INP: (int list/string) chanList = list of channels, or string '?'
     to get current list of channels, or string 'a' or 'al' or 'all'
     to set current list to all possible channels. Default: '?'
OUT: none
```

8.5.18 printCurrChanList(self):

printCurrChanList < DataEntity: < BoaDataEntity.py

```
DES: print the current channel list in somehow "clever" way
OUT: a string representing the current channel list
```

8.5.19 checkChanList(self,inList):

checkChanList < DataEntity: < BoaDataEntity.py

```
DES: Return a list of valid channels
INP: (int list/string) inList: list of channel numbers to get, or
     empty list to get the complete list of unflagged channels, or
     'all' or 'al' or 'a' to get the complete list of channels
OUT: (int list) list of channel numbers
```

8.5.20 getChanData(self,dataType='',chan=1,phase=0,flag=0):

getChanData < DataEntity: < BoaDataEntity.py

```
DES: get data of one channel
INP: (string) dataType = type of data
     (int)    chan = channel number
     (int)   phase = -1:all data; 0:phase difference; 1:phase 1; 2:phase
     (int)    flag = data flag to be used
OUT: (float) array = data of one channel
```

8.5.21 getChanListData(self,type='',chanList=[],phase=0,flag=0):

getChanListData < DataEntity: < BoaDataEntity.py

```
DES: get data for list of channels
INP: (string) type = type of data
     (int list) chan = channel list
     (int)   phase = -1:all data; 0:phase difference; 1:phase 1; 2:phase 2
     (int)    flag = data flag to be used
OUT: (float) array = data of the input list of channels
```

8.5.22 signal(self,chanList=[],phase=0,flag=0,\

signal < DataEntity: < BoaDataEntity.py

```
DES: plot time series of flux density
INP: (int list) chanList = list of channels
     (int)         phase = phase to be plotted
     (int)          flag = flag to be used
```

8.5.23 plotCorrel(self,chanRef=1,chanList=[],phase=0,flag=0,\

plotCorrel < DataEntity: < BoaDataEntity.py

```
DES: plot flux density of a list of channels vs. flux density of a
     reference channel
```

```
INP: (int)       chanRef = reference channel number
     (int list) chanList = list of channels
     (int)         phase = phase to be plotted
     (int)          flag = flag to be used
```

8.5.24 azimuth(self,chanList=[],phase=0,flag=0,\

azimuth < DataEntity: < BoaDataEntity.py

```
DES: plot time series of azimuth
INP: (int list) chanList = list of channels
     (int)         phase = phase to be plotted
     (int)          flag = flag to be used
```

8.5.25 elevation(self,chanList=[],phase=0,flag=0,\

elevation < DataEntity: < BoaDataEntity.py

```
DES: plot time series of elevation
INP: (int list) chanList = list of channels
     (int)         phase = phase to be plotted
     (int)          flag = flag to be used
```

8.5.26 azel(self,chanList=[],phase=0,flag=0,\

azel < DataEntity: < BoaDataEntity.py

```
DES: plot azimuth vs. elevation
INP: (int list) chanList = list of channels
     (int)         phase = phase to be plotted
     (int)          flag = flag to be used
```

8.5.27 eloff(self,chanList=[],phase=0,flag=0,\

eloff < DataEntity: < BoaDataEntity.py

```
DES: plot time series of azimuth offset
INP: (int list) chanList = list of channels
     (int)         phase = phase to be plotted
     (int)          flag = flag to be used
```

8.5.28 azoff(self,chanList=[],phase=0,flag=0,\

azoff < DataEntity: < BoaDataEntity.py

```
DES: plot time series of elevation offset
INP: (int list) chanList = list of channels
     (int)         phase = phase to be plotted
     (int)          flag = flag to be used
```

8.5.29 azeloff(self,chanList=[],phase=0,flag=0,\

azeloff < DataEntity: < BoaDataEntity.py

```
DES: plot elevation offset versus azimuth offset
INP: (int list) chanList = list of channels
     (int)         phase = phase to be plotted
     (int)          flag = flag to be used
```

8.5.30 plotMean(self,chanList=[],phase=0,flag=0,\

plotMean < DataEntity: < BoaDataEntity.py

```
DES: plot mean flux value vs. subscan number
WARNING: flag handling not implemented yet
INP: (int list) chanList = list of channels
     (int)         phase = phase to be plotted
     (int)          flag = flag to be used
```

8.5.31 plotRms(self,chanList=[],phase=0,flag=0,\

plotRms < DataEntity: < BoaDataEntity.py

```
DES: plot flux r.m.s. vs. subscan number
WARNING: flag handling not implemented yet
INP: (int list) chanList = list of channels
     (int)         phase = phase to be plotted
     (int)          flag = flag to be used
```

8.5.32 plotMeanChan(self,chanList=[],phase=0,flag=0,\

plotMeanChan < DataEntity: < BoaDataEntity.py

```
DES: PLotting the MEAN value for each subscan against channel number.
```

8.5.33 plotRmsChan(self,chanList=[],phase=0,flag=0,\

plotRmsChan < DataEntity: < BoaDataEntity.py

```
DES: PLotting the RMS value for each subscan against channel number.
```

8.5.34 chanMap(self,chanList=[],phase=0,flag=0,oversamp=2.):

chanMap < DataEntity: < BoaDataEntity.py

Attention: DES is missing! Please check the source code.
TXT:

```
NAM: chanMap (method)
```

8.5.35 slowChanMap(self,chanList=[],phase=0,flag=0,oversamp=2.,sizeX=[],sizeY=[]):

slowChanMap < DataEntity: < BoaDataEntity.py

```
NAM: slowChanMap (method)
DES: produce channel maps. Loop on pixels to reduce the size of allocated arrays
```

8.5.36 fastChanMap(self,chanList=[],phase=0,flag=0,oversamp=2.):

fastChanMap < DataEntity: < BoaDataEntity.py

Attention: DES is missing! Please check the source code.
TXT:

```
NAM: chanMap (method)
```

8.5.37 fft(self,chanList=[],phase=0,flag=0,\

fft < DataEntity: < BoaDataEntity.py

```
DES: plot FFT of signal
INP: (int list) chanList = list of channels
     (int)         phase = phase to be plotted
     (int)          flag = flag to be used
     (logical)  optimize = if set, limit the computation to a power of 2
                           elements
```

8.6 f1

f1 < BoaF1.f90

```
! NAM: f1 (module f90) in file BoaF1.f90
!
! DES: module collecting f90 subroutines not (yet) belonging to a particular class:
!        gsmooth
!        reimage
!        compress
!        get_subscan_index
! USE:
!      Call:  f90.f1.gsmooth(...)  etc. See examples in ftest.py
!      See BoaData.f90 for compilation instructions.
!! HIS:
!      FB040501  add and revise subroutines. data module reduced to constant attributes
!      FB040508  major revisions to some routines
!      FB040514  add  gsmooth
!      FB040514  corrections to gsmooth
!      FB040515  split f90 module into classes
!      FB040610  revise get_subscan_index: add second index for end index
```

History:

```
! NAM: f1 (module f90) in file BoaF1.f90
!
! DES: module collecting f90 subroutines not (yet) belonging to a particular class:
!        gsmooth
!        reimage
!        compress
!        get_subscan_index
! USE:
!      Call:  f90.f1.gsmooth(...)  etc. See examples in ftest.py
!      See BoaData.f90 for compilation instructions.
!! HIS:
!      FB040501  add and revise subroutines. data module reduced to constant attributes
!      FB040508  major revisions to some routines
!      FB040514  add  gsmooth
!      FB040514  corrections to gsmooth
```

```
!      FB040515  split f90 module into classes
!      FB040610  revise get_subscan_index: add second index for end index
```

8.6.1 gsmooth(image,fwhm,ii,jj)

gsmooth < f1 < BoaF1.f90

```
! NAM: gsmooth (subroutine)
! HIS: FB040514 created
! DES: Smooth an image with a Gaussian kernel
!        f(x) = exp( -4 ln2 [x/FWHM]^2 )  where 4*ln2 = 2.772588
!        f(FWHM/2) = 0.5,  f(FWHM) = 0.0625, f(1.5*FWHM)=0.002
!      that extends to a*FWHM, where a=1.5 (may be changed).
!      The kernel is in fact a square box.
!      For FWHM=1, it is of size 5x5, for 2 its 7x7, for 3 11x11 etc.
!      The computation of the kernel values is approximate, simply
!      taking f(x) at the pixel center.
!      Blanked image pixels are ignored in the averaging.
!      Current blanking value = -1.e-7
!      Blanks will be unblanked if they have nonblanked pixels in the
!      kernel range, else they stay at the blank value. Note that blanks
!      are not considered to have zero value!
! INP:
!      image : input rank-2 array('f') with bounds (ii,jj); ii>1,jj>1
!      fwhm  : input float. FWHM of Gaussian in pixel.
! OUT:
!      image : rank-2 array('f') with bounds (ii,jj).
!
! USE: image = f90.f1.gsmooth(image,fwhm)
!      EXAMPLE:
!        from Numeric import *
!        import f90,time
!        im = zeros(21,21,'f') +5.
!        im[11,11] = 10.      # make a peak in the center
!        im[11,10] = -1.e-7   # set to blanking value
!        t0 = time.clock()
!        im = f90.f1.gsmooth(im,3.)
```

```
!        print str(time.clock()-t0)
!
!      for a 1001^2 array with FWHM=3 (kernel size=11x11) it takes 3.2 sec
!                                  =5              17x17           5.6
```

8.6.2 reimage(image,data,xy,dxy,zero_offset,xy_scale,ii,jj,kk)

reimage < f1 < BoaF1.f90

```
! NAM: reimage (subroutine)
! HIS: FB040510 created
! DES: fill an image array with data in pixel boxes of given size
!        image = reimage(image,data,xy,dxy,zero_offset,xy_scale)
! INP:
!        image       : input rank-2 array('f') with bounds (ii,jj)
!        data        : input rank-1 array('f') with bounds (kk)
!        xy          : input rank-2 array('f') with bounds (2,kk)
!        dxy         : input rank-2 array('f') with bounds (2,kk)  -- half width!
!        zero_offset : input rank-1 array('f') with bounds (2)
!                      bottom left corner user coordinate (eg arcsec) offset.
!        xy_scale    : input rank-1 array('f') with bounds (2)
!                      image scale, pixel per user coordinate (eg pix/arcsec)
! OUT:
!        image       : rank-2 array('f') with bounds (ii,jj)
! USE: Example:
!        from Numeric import *
!        from fortran import f90
!        im = zeros((11,11),'f')  # odd size since coordinates refer to pix center
!        data    = array([10,20,30],'f')
!        xyscale = array([0.1,0.1],'f')
!        xy      = array([ [-50,0,20],[-30,0,20] ],'f')
!        dxy     = array([ [11,6,11] , [11,6,11] ],'f')
!        zerooffset = array([-50,-50],'f') # blc pix refers to -50,-50
!        im = f90.f1.reimage(im,data,xy,dxy,zerooffset,xyscale)
!        print im
```

8.6.3 compress(xdata,xflag,flag_value,ii)

compress < f1 < BoaF1.f90

```
! NAM: compress (subroutine)
! DES: compress array based on mask
! INP: data_input (f): 1-D array with data to be returned where flag = flag_value
!      flag       (i): 1-D array, must be same size as data_input
!      flag_value (i): select data where flag=flag_value
! OUT: tuple (x,ii): 1-D array of same size as data_input where first n elements are
!                    those of data_input where flag = flag_value
! HIS: FB040430 created
! USE:
!      Unfortunately it is not possible to pass from fortran to python an output
!      array the size of which is computed in fortran. Therefore we must truncate
!      the returned array at the size of the selected true mask elements.
!
!      Example:
!            input_array = array(range(5),'f')
!            flag_array  = array([0,1,0,1,0],'i')
!            compress_array,nmax = f90.f1.compress(input_array,flag_array,1)
!            compress_array = compress_array[0:nmax]
!            print "nmax compressed:",nmax,compress_array
```

8.6.4 get_subscan_index(xsubscan_index,ii)

get_subscan_index < f1 < BoaF1.f90

```
! NAM: get_subscan_index (subroutine)
! DES: get the beginning index of each subscan data in phase array
!      by comparing the subscan begin time with lst_p.
!      Result must be fetched from f90.data module.
!      Might be good to specify the ending index as well if we sample
!      data in between subscans.
! INP: data.subscan_time   f
!      data.lst_p          f
! OUT: data.subscan_index  i
```

```
! HIS: FB040429 created
!      FB040610 add second index to xsubscan_index
```

8.7 Focus:

Focus: < BoaFocus.py

```
NAM: Focus (class)
VER: 0.15 22.05.2004
DES: An object of this class is responsible for the focus reduction
     of single or multiple scans and provides the offsets.
```

History:

```
NAM: BoaFocus.py (file)
VER: 0.38 17.09.2004
DES: contains the BoA focus class
HIS: MA040112  initial version
     SM040511  new version
     SM040512  new version
     SM040513  new version
     SM040514  with Marcus Albrecht: first runnig version
     MA040519  included Bogli plotting
     SM040522  corrected error in determination of the errors
               changed from simple error to confidence interval
     SM040712  minor changes
     SM040713  added method to derive and show focus results from
               accumulated/multiple scans, initial version
     SM040714  added method to derive and show focus results from
               accumulated/multiple scans, initial version
     MA040716  added accumulated focus
     SM040910  small changes
     SM040913  changes on accumulated focus
     SM040915  changes on accumulated focus
     SM040916  changes on accumulated focus
     SM040917  changes on accumulated focus
```

8.7.1 __init__(self,BoaB):

__init__ < Focus: < BoaFocus.py

```
DES: Initialise an instance
INP: (object) BoaB = BoA business object
HIS: MA040405  initial version
     SM040514  with Marcus Albrecht: first runnig version
```

8.7.2 scanFocus(self,chan=-1,phase=0,flag=0,plot=1,omit=[],r=0):

scanFocus < Focus: < BoaFocus.py

```
NAM: scanFocus (method)
DES: focus method for single scans
INP: (i)      chan  = pointing channel
     (i)      phase = phase to be used
     (i)      flag  = flag to be used (0=unflagged)
     (l)      plot  = 1: provide graphical output
     (i list) omit  = list of observations (=subscans) to be omitted
HIS: MA040405  initial version
     SM040514  with Marcus Albrecht: first runnig version
```

8.7.3 accFocus(self,com='?',r=0):

accFocus < Focus: < BoaFocus.py

```
NAM: scanFocus (method)
DES: focus method for single scans
INP: (i)      chan  = pointing channel
     (i)      phase = phase to be used
     (i)      flag  = flag to be used (0=unflagged)
     (l)      plot  = 1: provide graphical output
     (i list) omit  = list of observations (=subscans) to be omitted
HIS: MA040405  initial version
     SM040514  with Marcus Albrecht: first runnig version
```

8.7.4 detSubStruct(self,chan=1,flag=0):

detSubStruct < Focus: < BoaFocus.py

```
NAM: detSubStruct (method)
DES: determine the structure of the scan = first and last index
     of each observation (=subscan)
INP: (int) chan = channel
     (int) flag = flag to be used (0=unflagged)
HIS: MA040405  initial version
     SM040514  with Marcus Albrecht: first runnig version
```

8.7.5 detScanExtr(self):

detScanExtr < Focus: < BoaFocus.py

```
NAM: detExtr (method)
DES: determine extrema of scan data
HIS: MA040405  initial version
     SM040916  small changes
```

8.7.6 createModelData(self):

createModelData < Focus: < BoaFocus.py

```
NAM: createModelData (method)
DES: create data for graphical display of fit
HIS: MA040405  initial version
     SM040915  small changes
     SM040917  small changes
```

8.7.7 printScanResults(self):

printScanResults < Focus: < BoaFocus.py

```
NAM: printScanResults (method)
DES: print results for scan
HIS: MA040405  initial version
     SM040514  with Marcus Albrecht: first runnig version
     SM040522  new error calculation
     SM040910  small changes
     SM040915  small changes
```

8.7.8 initAccFocus(self):

initAccFocus < Focus: < BoaFocus.py

```
DES: Initialise an instance
HIS: MA040716  initial version
     SM040913  small changes
     SM040915  updated
```

8.7.9 addToAccFocus(self):

addToAccFocus < Focus: < BoaFocus.py

```
DES: Add a scan to accumulated focus
HIS: MA040716  initial version
     SM040913  small changes
     SM040917  updated
```

8.7.10 detAccExtr(self):

detAccExtr < Focus: < BoaFocus.py

```
DES: Add a scan to accumulated focus
HIS: MA040716  initial version
     SM040913  small changes
     SM040917  updated
```

8.7.11 createAccModelData(self):

createAccModelData < Focus: < BoaFocus.py

```
DES: Creates accumulated focus model data
HIS: MA040716  initial version
     SM040913  small changes
     SM040916  updated
     SM040917  small changes
```

8.7.12 printAccResults(self):

printAccResults < Focus: < BoaFocus.py

```
DES: Creates accumulated focus model data
HIS: MA040716  initial version
     SM040913  small changes
     SM040916  updated
     SM040917  small changes
```

8.8 baseline

baseline < BoaFortranBaseLine.f90

```
! NAM: baseline (module f90) in file BoaFortranBaseLine.f90
!
! DES: module collecting f90 subroutines:
!      polyfit
!      addpoly
!
! USE: Call:  f90.baseline.polyfit(...)  etc. See examples in ftest.py
!
! HIS: FB040426  assemble flag,mean_rms,polyfit in one module
!      FB040501  add and revise subroutines. data module reduced to constant attributes
!      FB040508  major revisions to some routines
!      FB040515  split f90 module into classes
!      SM041109  renamed module to avoid conflicts with py module
```

History:


```
! NAM: baseline (module f90) in file BoaFortranBaseLine.f90
!
! DES: module collecting f90 subroutines:
!      polyfit
!      addpoly
!
! USE: Call:  f90.baseline.polyfit(...)  etc. See examples in ftest.py
!
! HIS: FB040426  assemble flag,mean_rms,polyfit in one module
!      FB040501  add and revise subroutines. data module reduced to constant attributes
!      FB040508  major revisions to some routines
!      FB040515  split f90 module into classes
!      SM041109  renamed module to avoid conflicts with py module
```

8.9 Image:

Image: < BoaMapping.py

```
NAM: Image (class)
DES: An object of this class describe an image an it axis
```

History:

```
NAM: BoaMapping.py (module)
DES: contains the BoAMapping and Image classes
```

8.9.1 wcs2pix(self,X,Y):

wcs2pix < Image: < BoaMapping.py

```
DES: Convert from physical coordinate describe by self.WCS to pixel coordinate
INP: float (X,Y) : the physical coordinate to convert from
OUT: float (i,j) : the pixel coordinate(s)
```

8.9.2 wcs2phy(self,i,j):

wcs2phy < Image: < BoaMapping.py

```
DES: Convert from pixel coordinates to physical (world) coordinates
INP: float (i,j) : the pixel coordinate to convert from
OUT: float (X,Y) : the physical coordinate
```

8.9.3 computeWCS(self,pixelSize,sizeX=[],sizeY=[],minmax=[]):

computeWCS < Image: < BoaMapping.py

```
DES: fill main WCS keywords according to pixel size and map limits
INP: (int) pixelSize = size of pixel in acrsecond
     (float) sizeX = map limits in azimuth, in arcsecond
     (float) sizeY = map limits in elevation, in arcsecond
     (float) minmax = [minAzoff,maxAzoff,minEloff,maxEloff] in this order
```

8.9.4 physicalCoordinates(self):

physicalCoordinates < Image: < BoaMapping.py

```
DES: return arrays with physical units corresponding to the map
```

8.9.5 display(self,rms=0,weight=0,\

display < Image: < BoaMapping.py

```
DES: show the reconstructed maps in (Az,El)
INP: (boolean) rms,weight    : plot the rms or weight map instead of signal map
     (string)  style         : the style used for the color (default g2r)
     (string) labelX, labelY : the X and Y labels
     (boolen) wedge          : draw a wedge ? (default : yes)
```

8.10 Map(BoaDataAnalyser.DataAna):

Map(BoaDataAnalyser.DataAna): < BoaMapping.py

```
NAM: Map (class)
DES: An object of this class is responsible for the restoration of
     mapping data of single or multiple files.
```

8.10.1 __init__(self):

__init__ < Map(BoaDataAnalyser.DataAna): < BoaMapping.py

```
DES: Initialise an instance.
```

8.10.2 slowMap(self,chanList=[],phase=0,flag=0,oversamp=2.0,beammap=0,\

slowMap < Map(BoaDataAnalyser.DataAna): < BoaMapping.py

```
DES: reconstruct a map in (Az,El) coordinates combining bolometers
        INP: (int list) chanList = channels to consider
             (int) phase = phase to plot
             (int) flag  = flag values to consider
             (float) oversamp = oversampling factor (beam fwhm / pixel size). Default=2.
             (list float) sizeX = limits in Az of the map
             (list float) sizeY = limits in El of the map
             (list float) offsets = Az,El  to recenter a point source
             (logical) noNan = remove NaN in self.Results?
             (str) style = color table to use in image
```

8.10.3 slowMap2(self,chanList=[],phase=0,flag=0,oversamp=2.0,beammap=0,\

slowMap2 < Map(BoaDataAnalyser.DataAna): < BoaMapping.py

```
DES: reconstruct a map in (Az,El) coordinates combining bolometers
        INP: (int list) chanList = channels to consider
             (int) phase = phase to plot
             (int) flag  = flag values to consider
             (float) oversamp = oversampling factor (beam fwhm / pixel size). Default=2.
             (list float) sizeX = limits in Az of the map
             (list float) sizeY = limits in El of the map
             (list float) offsets = Az,El  to recenter a point source
             (logical) noNan = remove NaN in self.Results?
             (str) style = color table to use in image
```

8.10.4 fastMap(self,chanList=[],phase=0,flag=0,oversamp=2.0,beammap=0,\

fastMap < Map(BoaDataAnalyser.DataAna): < BoaMapping.py

```
DES: reconstruct a map in (Az,El) coordinates combining bolometers
        INP: (int list) chanList = channels to consider
             (int) phase = phase to plot
             (int) flag  = flag values to consider
             (float) oversamp = oversampling factor (beam fwhm / pixel size). Default=2.
             (list float) sizeX = limits in Az of the map
             (list float) sizeY = limits in El of the map
             (list float) offsets = Az,El  to recenter a point source
             (logical) noNan = remove NaN in self.Results?
             (str) style = color table to use in image
             (logical) noSmooth = do not smooth with beam?
```

8.10.5 showMap(self,style='g2r',labelX="\ gDAz["]",labelY="\ gDEl["]",wedge=1,limitsZ=[]):

showMap < Map(BoaDataAnalyser.DataAna): < BoaMapping.py

```
DES: show the reconstructed map in (Az,El)
```

8.10.6 beamMap(self,chanList=[],phase=0,flag=0,oversamp=2.0,sizeX=[],sizeY=[]):

beamMap < Map(BoaDataAnalyser.DataAna): < BoaMapping.py

```
DES: build a beam map in (Az,El) coordinates
        INP: (int list) chanList = channels to consider
             (int) phase = phase to plot
             (int) flag  = flag values to consider
             (float) oversamp = oversampling factor (beam fwhm / pixel size). Default=2.
             (list float) sizeX = limits in Az of the map
             (list float) sizeY = limits in El of the map
```

8.10.7 fastBeam(self,chanList=[],phase=0,flag=0,oversamp=2.0,\

fastBeam < Map(BoaDataAnalyser.DataAna): < BoaMapping.py

```
        DES: build a beam map in (Az,El) coordinates summing up channel maps
        INP: (int list) chanList = channels to consider
             (int) phase = phase to plot
             (int) flag  = flag values to consider
             (float) oversamp = oversampling factor (beam fwhm / pixel size). Default=2.
             (list float) sizeX = limits in Az of the map
             (list float) sizeY = limits in El of the map
             (str) style = name of color map to display map
             (log) wedge = plot a wedge?
```

8.10.8 wcs2pix(self,X,Y):

wcs2pix < Map(BoaDataAnalyser.DataAna): < BoaMapping.py

```
        DES: Convert from physical coordinate describe by self.WCS to pixel coordinate
        INP: float (X,Y) : the physical coordinate to convert from
        OUT: float (i,j) : the pixel coordinate(s)
```

8.10.9 wcs2phy(self,i,j):

wcs2phy < Map(BoaDataAnalyser.DataAna): < BoaMapping.py

```
DES: Convert from pixel coordinates to physical (world) coordinates
INP: float (i,j) : the pixel coordinate to convert from
OUT: float (X,Y) : the physical coordinate
```

8.10.10 computeWCS(self,pixelSize,sizeX=[],sizeY=[],minmax=[]):

computeWCS < Map(BoaDataAnalyser.DataAna): < BoaMapping.py

```
DES: fill main WCS keywords according to pixel size and map limits
INP: (int) pixelSize = size of pixel in acrsecond
     (float) sizeX = map limits in azimuth, in arcsecond
     (float) sizeY = map limits in elevation, in arcsecond
     (float) minmax = [minAzoff,maxAzoff,minEloff,maxEloff] in this order
```

8.10.11 boloPos(self,chanList,allFlux):

boloPos < Map(BoaDataAnalyser.DataAna): < BoaMapping.py

```
DES: return the offset w.r.t reference channel and the flatfielded flux
INP: (list) chanList : list of channel to take into account
     (array) allFlux : correctonding fluxes
```

8.10.12 getPixel(self,nbPix=3):

getPixel < Map(BoaDataAnalyser.DataAna): < BoaMapping.py

```
DES: allow user to get pixel values using mouse
INP: (int)  nbPix : size of area to compute average
```

8.11 MessHand:

MessHand: < BoaMessageHandler.py

```
NAM: MessHand (class)
DES: An object of this class is responsible for the management of output
     messages as well as the creation of message files.
```

History:

```
NAM: BoaMessageHandler.py (module)
DES: contains the BoA message handler class
```

8.11.1 __init__(self,logName='Unknown'):

__init__ < MessHand: < BoaMessageHandler.py

```
DES: initialise an instance
INP: -
OUT: -
```

8.11.2 Welcome(self):

Welcome < MessHand: < BoaMessageHandler.py

```
DES: print welcome message
INP: -
OUT: screen output
```

8.11.3 setMaxWeight(self,weight='2'):

setMaxWeight < MessHand: < BoaMessageHandler.py

```
DES: Set the maximum weight of messages to be printed.
INP: (int) weight = maximum weight

             1: errors, queries
             2: warnings
             3: short info
             4: extended info
             5: debug
```

8.11.4 error(self,message=''):

error < MessHand: < BoaMessageHandler.py

```
DES: to print an error message
INP: (sring) message
```

8.11.5 warning(self,message=''):

warning < MessHand: < BoaMessageHandler.py

```
DES: to print an warning message
INP: (sring) message
```

8.11.6 info(self,message=''):

info < MessHand: < BoaMessageHandler.py

```
DES: to print an info message
INP: (sring) message
```

8.11.7 longinfo(self,message=”):

longinfo < MessHand: < BoaMessageHandler.py

```
DES: to print an long info message
INP: (sring) message
```

8.11.8 debug(self,message=”):

debug < MessHand: < BoaMessageHandler.py

```
DES: to print an debug message
INP: (sring) message
```

8.11.9 setMess(self,weight=1,message=”):

setMess < MessHand: < BoaMessageHandler.py

```
DES: deposit messages for screen output and message files
INP: (int) weight = weight of transferred message (see setMaxWeight)
     (string) message = message to be printed and added to message file
OUT: -
```

8.11.10 initMessFile(self,filename=”boa.mes”):

initMessFile < MessHand: < BoaMessageHandler.py

```
DES: set & initialise new message file
INP: -
OUT: screen output
```

8.11.11 closeMessFile(self):

closeMessFile < MessHand: < BoaMessageHandler.py

```
DES: set self.__existMessFile to 0 and file name to ""
INP: -
OUT: -
```

8.12 OnOff:

OnOff: < BoaOnOff.py

```
NAM: OnOff (class)
VER: 0.1 12.01.2004
DES: An object of this class provides methods for the onoff reduction
     of single or multiple scans.
```

History:

```
NAM: BoaOnOff.py (file)
VER: 0.1 12.01.2004
DES: contains the BoA onoff class
HIS: 12.01.2004  M. Albrecht      Initial version
```

8.12.1 __init__(self,BoaB):

__init__ < OnOff: < BoaOnOff.py

```
DES: Initialise an instance
INP: (object) BoaB = BoA business object
```

8.13 Point(BoaMapping.Map):

Point(BoaMapping.Map): < BoaPointing.py

```
NAM: Point (class)
DES: An object of this class is responsible for the reduction of
     pointing scan(s)
```

History:

```
NAM: BoaPoint.py (module)
DES: contains the BoA Pointing reduction tools
```

Additional information:
Example: In the figures below the graphical output of a pointing of a single scan (8.1) and of accumulated scans (8.2) is shown.

Levenberg-Marquardt:

The Levenberg-Marquardt algorithm can be thought of as a trust-region modification of the Gauss-Newton algorithm. Levenberg-Marquardt steps s_k are obtained by solving subproblems of the form

$$\min \left\{ \frac{1}{2} \left\| f'(x_k)s + f(x_k) \right\|_2^2 : \|D_{k^s}\|_2 \leq \Delta_k \right\} \tag{8.1}$$

for some $\Delta_k > 0$ and scaling matrix Δ_k. The trust-region radius is adjusted between iterations according to the agreement between predicted and actual reduction in the objective function. For a step to be accepted, the ratio

$$\rho_k = \frac{r(x_k) - r(x_k + s_k)}{r(x_k) - \frac{1}{2} \|f'(x_k)s + f(x_k)\|_2^2 : \|D_{k^s}\|_2} \tag{8.2}$$

must exceed a small positive number.(typically .0001). If this test is failed, the trust region is decreased and the step is recalculated. When the ratio is close to one, the trust region for the next iteration is expanded.
Levenberg-Marquardt codes usually determine the step by noting that the solution of 8.1 also satisfies the equation

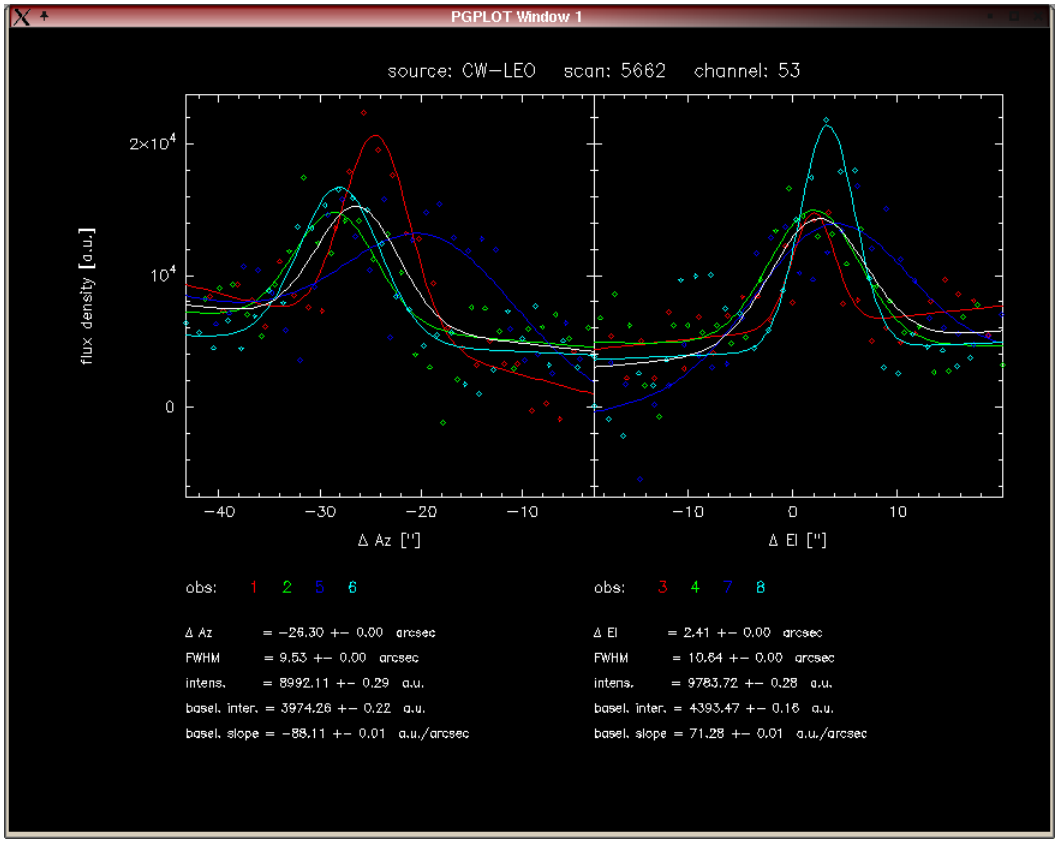
PGPLOT Window 1
source: CW-LEO scan: 5662 channel: 53
flux density [a.u.]
2×10^4
10^4
0
−40 −30 −20 −10
Δ Az ["]
−10 0 10
Δ El ["]
obs: 1 2 5 6
Δ Az = −26.30 +− 0.00 arcsec
FWHM = 9.53 +− 0.00 arcsec
intens. = 8992.11 +− 0.29 a.u.
basel. inter. = 3974.26 +− 0.22 a.u.
basel. slope = −88.11 +− 0.01 a.u./arcsec
obs: 3 4 7 8
Δ El = 2.41 +− 0.00 arcsec
FWHM = 10.64 +− 0.00 arcsec
intens. = 9783.72 +− 0.28 a.u.
basel. inter. = 4393.47 +− 0.16 a.u.
basel. slope = 71.28 +− 0.01 a.u./arcsec

Figure 8.1: Graphical output of a pointing of a single scan of a point source, colors represent different scans, symbols are measured values, drawn lines are Levenberg-Marquardt fits.

$$\left(f'(x_k)^T\, f'(x_k) + \lambda_k\, D_k^T\, D_k\right)\, s_k = -f'(x_k)^T\, f'(x_k) \tag{8.3}$$

for some $\lambda_k \geq 0$.
The Lagrange multiplier λ_k is zero if the minimum-norm Gauss-Newton step is smaller than Δ_k; otherwise λ_k is chosen so that $\|\Delta_k\, s_k\|_2 = \Delta_k$.
Equations 8.3 are simply the normal equations for the least squares problem

$$\min\left\{\left\|\begin{bmatrix} f'(x_k) \\ \lambda_k^{1/2}\, D_k \end{bmatrix} s + \begin{bmatrix} f(x_k) \\ 0 \end{bmatrix}\right\|_2^2 : s \in \mathbb{R}^n\right\} \tag{8.4}$$

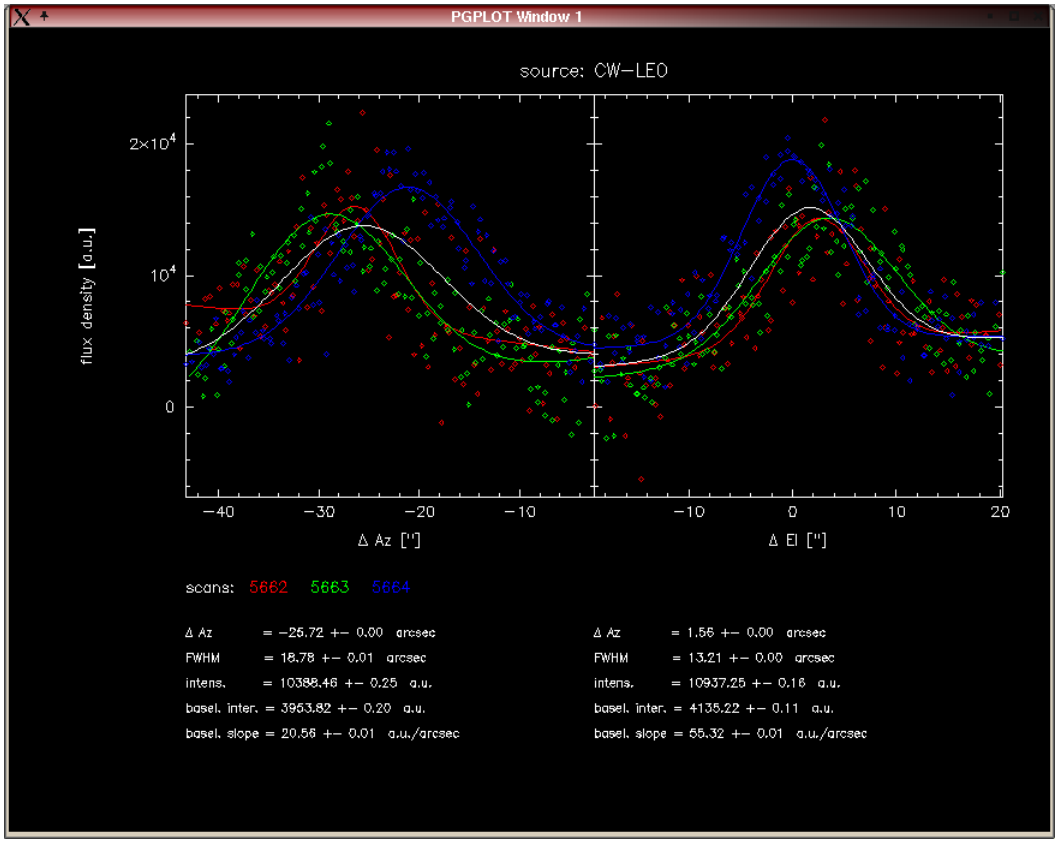

Figure 8.2: Graphical output of a pointing of accumulated scans of a point source, colors represent different scans, symbols are measured values, drawn lines are Levenberg-Marquardt fits.

Efficient factorization of the coefficient matrix in 8.4 can be performed by a combination of Householder and Givens transformations. The Levenberg-Marquardt algorithm has proved to be an effective and popular way to solve nonlinear least squares problems.

8.13.1 __init__(self):

__init__ < Point(BoaMapping.Map): < BoaPointing.py

```
DES: Initialise an instance.
```

8.13.2 iterMap(self,chanList=[],phase=0,flag=0,sizeX=[],sizeY=[]):

iterMap < Point(BoaMapping.Map): < BoaPointing.py

```
DES: reconstruct a map in (Az,El) coordinates combining bolometers
and using varying scale to zoom on signal
        INP: (int list) chanList = channels to consider
             (int) phase = phase to plot
             (int) flag  = flag values to consider
             (list float) sizeX = limits in Az of the map
             (list float) sizeY = limits in El of the map
```

8.13.3 solvePointing(self,chanList=[],gradient=1,radius=0):

solvePointing < Point(BoaMapping.Map): < BoaPointing.py

```
        DES: compute the offset
        INP: (int list) chanList: list of channels to be used (default: all)
             (boolean)  gradient: shall we fit a gradient ? (default: yes)
             (float)    radius  : use only bolo inside this radius (default 0 : all)
```

8.13.4 showPointing(self,plot=1):

showPointing < Point(BoaMapping.Map): < BoaPointing.py

```
        DES: compute the offset
        INP: (logical) plot: if True, also display the results on a map
```

8.14 snf

snf < BoaSNF.f90

```
! NAM: snf (module f90) in file BoaSNF.f90
!
! DES: module collecting f90 subroutines for sky noise subtraction:
!        corfit
!        cmatrix
!
! USE: f90.snf.corfit(...)  etc. See examples in TestFB.py
!      For compilation instruction see BoaData.f90
!
! HIS: FB040531 consolidate cmatrix and corfit in this new module
!      FB040601 fix cmatrix
```

History:


```
! NAM: snf (module f90) in file BoaSNF.f90
!
! DES: module collecting f90 subroutines for sky noise subtraction:
!        corfit
!        cmatrix
!
! USE: f90.snf.corfit(...)  etc. See examples in TestFB.py
!      For compilation instruction see BoaData.f90
!
! HIS: FB040531 consolidate cmatrix and corfit in this new module
!      FB040601 fix cmatrix
```

8.15 Sky:

Sky: < BoaSkydip.py

```
NAM: Sky (class)
VER: 0.1 12.01.2004
DES: An object of this class provides methods to conduct the
     determination of the opacity of single or multiple scans.
```

History:

```
NAM: BoaSkydip.py (file)
VER: 0.1 12.01.2004
DES: contains the BoA skydip class
HIS: 12.01.2004  M. Albrecht      Initial version
```

8.15.1 __init__(self,BoaB):

__init__ < Sky: < BoaSkydip.py

```
DES: Initialise an instance.
INP: (object) BoaB = BoA business object
```

8.16 test1

test1 < BoaTest1.f90

```
! NAM: test1 (module f90) in file BoaTest1.f90
! DES: module collecting test subroutines
! USE: wrap:  f2py -c -m f90 BoaData.f90 BoaTest1.f90 BoaTest2.f90  ...
!      call:  f90.test1.flag(...)  etc.
! HIS: FB040303  change to _p
```

History:

```
! NAM: test1 (module f90) in file BoaTest1.f90
! DES: module collecting test subroutines
! USE: wrap:  f2py -c -m f90 BoaData.f90 BoaTest1.f90 BoaTest2.f90  ...
!      call:  f90.test1.flag(...)  etc.
! HIS: FB040303  change to _p
```

8.16.1 flag(n_ch,n_ph,low,high)

flag < test1 < BoaTest1.f90

```
! NAM: test1.flag  (sub)
! VER: 9.12.03 f.bertoldi initial
!      9.02.04 fb now flag outside range
! DES: only for testing and illustration
!      set flag for given channel if data <low or >high
! INP: channel = channel number for which data to be flagged
!      low     = lower limit
!      high    = upper limit
! OUT:
! HIS: FB040303  change to _p
```

8.17 MamboMBFits:

MamboMBFits: < MamboMBFits.py

```
  DES: Objects of this class contain two attributes of the FitsFile
  class. The 'Mambo' attribute stores the data as they appear in a
  MAMBO FITS file, and the 'MBfits' attribute contains the data in
  the MB-FITS format. Methods to do conversions in both directions
  are provided by this class.
```

History:

```
NAM: MamboMBFits.py (file)
DES: this module defines the MamboMBFits class, which contains both
a Mambo fits file  and an MB-Fits file, and  provides methods to do
conversions between these two formats.
VER: 24.05.2004
HIS: 17.02.2004  F. Schuller      Initial version
     17.02.2004 Initial version, from merging  of the mambo2MBFits
                and MB2MamboFits modules

  17-18.02.2004 Complete  restructuration  of the code  to a  more
                object-oriented style. Definition of a MamboMBFits
                class that contains two attributes of the FitsFile
                class, and  provides methods to  do conversions in
                both directions. The code  to do these conversions
                has been split to many sub-functions.
     27.02.2004 More stuff related to coordinates is now converted
                from Mambo to MB-FITS
     01.03.2004 Same  stuff (Astronomical  basis  frame) converted
                from MB-FITS to Mambo
     09.03.2004 More stuff converted.  Changes (also in Mambo.xml)
                to allow  the use of 'Ok'  rather than 'None'  for
                keywords that are "manually" processed
     10.03.2004 Implemented the convertFebepar method
     11.03.2004 More keywords converted: FREQUENC, TELSIZM...
     18.03.2004 Add conversion of  Fast Scanning observations from
                MAMBO to MB-FITS
     19.03.2004 The same for converting from MB-FITS to MAMBO
```

```
06.05.2004 Adapted for use within BoA
12.05.2004 Some small corrections for BoA
24.05.2004 Some corrections to work with real APEX files
   09.2004 a few more corrections after observing at APEX
FS050202   minor modif. for compatibility with APEX control PC
```

8.17.1 __init__(self,mamboName,mbName):

__init__ < MamboMBFits: < MamboMBFits.py

```
DES: Instanciation of a new MamboMBFits object.
INP: (str) mamboName = name of the Mambo file
     (str) mbName = name of the MB-FITS file
     in both cases, the .fits extension is appended if not present
HIS: FS041210  also handle names that already contain .fits
```

8.17.2 convertMambo2MBFits(self):

convertMambo2MBFits < MamboMBFits: < MamboMBFits.py

```
DES: This function reads in the content of a Mambo FITS file,
and writes out the data and associated parameters to a file
conforming to the MB-FITS format.
```

8.17.3 convertMB2MamboFits(self):

convertMB2MamboFits < MamboMBFits: < MamboMBFits.py

```
DES: This function reads in the content of an MB-FITS file,
and writes out the data to a file in the Mambo-FITS format,
and the associated parameters to a RCP file.
```

8.17.4 readMambo(self):

readMambo < MamboMBFits: < MamboMBFits.py

```
DES: This fills the TableList in the Mambo attribute with the
content of the MAMBO FITS file.
```

8.17.5 readMBfits(self):

readMBfits < MamboMBFits: < MamboMBFits.py

```
DES: This fills the TableList in the MBfits attribute with the
content of the MB-FITS file.
```

8.17.6 initMambo(self):

initMambo < MamboMBFits: < MamboMBFits.py

```
DES: Create a file in the Mambo FITS format. This generates only the
Primary header and subscan table, because the number of feeds must be
written in the Primary header before the data table is created.
```

8.17.7 initMB(self):

initMB < MamboMBFits: < MamboMBFits.py

```
DES: This generates the first three tables in the MB-FITS file
(Primary header, SCAN-MBFITS and FEBEPAR-MBFITS tables).
```

8.17.8 convertMamboPrimary(self):

convertMamboPrimary < MamboMBFits: < MamboMBFits.py

```
DES: This generates in the MB-FITS file all the header keywords that
have one direct equivalent in the MAMBO primary header.
```

8.17.9 fillMamboPrimary(self):

fillMamboPrimary < MamboMBFits: < MamboMBFits.py

```
DES: Update the keyword values in the Mambo Primary header,
using the equivalent keywords found in the MB-FITS file.
```

8.17.10 fillFebepar(self):

fillFebepar < MamboMBFits: < MamboMBFits.py

```
DES: This method writes one row in the FEBEPAR-MBFITS table, by
reading the content of a Mambo RCP file, specified by the total
number of pixels (40 or 120).
```

8.17.11 convertFebepar(self):

convertFebepar < MamboMBFits: < MamboMBFits.py

```
DES: This method generates a RCP file, where the relative gains
and offsets of pixels are stored, from the content of the FEBEPAR
table. The output file name is <FE_name>.rcp.
```

8.17.12 processMamboData(self):

processMamboData < MamboMBFits: < MamboMBFits.py

```
DES: This method generates ARRAYDATA, DATAPAR and MONITOR tables in
the MB-FITS file for every subscan in the MAMBO file.
```

8.17.13 fillMamboData(self):

fillMamboData < MamboMBFits: < MamboMBFits.py

```
DES: This methods fills the subscans and data tables in an output MAMBO
file, using the data previously read in from an MB-FITS file.
```

8.17.14 getScanType(self):

getScanType < MamboMBFits: < MamboMBFits.py

```
DES: Convert the informations about the observing type from
Mambo format (contained in OBSMODE + SRP1FLAG + SRP2FLAG) to
an MB-FITS SCANTYPE. Also return the SCANMODE, SCANGEOM,
SWTCHMOD, WOBSW, OBSTYPE and SCANDIR infos. The results are
stored in the Scantype attribute, which is a Dictionary.
```

8.17.15 getObsMode(self):

getObsMode < MamboMBFits: < MamboMBFits.py

```
DES: Convert the scanType (from MB-FITS) to OBSMODE + SRP1FLAG
+ SRP2FLAG as defined in the MAMBO format.
```

8.17.16 readRCP(self):

readRCP < MamboMBFits: < MamboMBFits.py

```
DES: Read the Receiver Channels Parameters from the file
'MRT_2002a_120.rcp' if the number of bolometers is 120, or
'MRT_2002s2_40.rcp' if it is 40. These files are supposed
to be in the local directory. Returns a list of tuples,
where each tuple is (Gain,X_off,Y_off).
```

8.17.17 sbas2ctype(self):

sbas2ctype < MamboMBFits: < MamboMBFits.py

```
DES: Converts the SBAS value to the MB-FITS keywords that define the
Astronomical basis frame: CTYPEj, WCSNAME, RADESYS, EQUINOX, and
eventually MOVEFRAM. All these are stored in a dictionary.
```

8.17.18 ctype2sbas(self):

ctype2sbas < MamboMBFits: < MamboMBFits.py

```
DES: Converts the infos about the Astronomical basis frame from MB-FITS
keywords (CTYPEj, EQUINOX...) to a SBAS value (+ epoch). The results
are stored and returned in a dictionary.
```

8.18 Timing:

Timing: < Utilities.py

```
NAM: Timing (class)
DES: easily profile time computation in program
```

History:

```
NAM: Utilities.py (module)
DES: contains utilities & methods for fitting functions
```

8.18.1 safeExp(x):

safeExp < Timing: < Utilities.py

```
NAM: safeExp (function)
DES: correct a bug in Numerical that raise an expection when
     computing exponential of small numbers, this take a lot of time !
     but faster thant converting to nummarray compute the exp and back to numeric !!
```

8.18.2 fitParabola(x,y,err):

fitParabola < Timing: < Utilities.py

```
NAM: fitParabola (method)
DES: fits parabola to the data using mpfit
INP: (float) x = x data
     (float) y = y data
```

8.18.3 detStartParaParabola(x,y):

detStartParaParabola < Timing: < Utilities.py

```
NAM: defStartParaParabola (method)
DES: define the proper start parameter to fit a parabola
INP: (float) x = x data
     (float) y = y data
```

8.18.4 parabola(p,fjac=None,x=None,y=None,err=None):

parabola < Timing: < Utilities.py

```
NAM: parabola
DES: function used by mpfit to fit a parabola
```

8.18.5 modelparabola(p,x):

modelparabola < Timing: < Utilities.py

```
NAM: modelparabola
DES: compute a model parabola at position x for a given set of parameters p
```

8.18.6 draw2Dgauss(p):

draw2Dgauss < Timing: < Utilities.py

```
NAM: draw2Dgauss
DES: return the coordinates of the 2*fwhm ellipse
     given the parameters p of a 2Dgaussian
```

8.18.7 modelBase2Dgauss(p,position):

modelBase2Dgauss < Timing: < Utilities.py

```
NAM: model2Dgauss
DES: compute a 2D gaussian defined by the parameter p wihtin the position
     position should be a list of 2 arrays of same dimensions defining the map
```

8.18.8 base2DGauss(p,fjac=None,x=None,y=None,err=None):

base2DGauss < Timing: < Utilities.py

```
NAM: base2DGauss
DES: function used by mpfit to fit a 2D gaussian+base
     (5 elmts array) p : parameters of the gaussian (see modelBase2Dgauss)
     (2d array)      x : position of  the pixels on the map
                         "x" = x[0] and "y" = x[1]
     (2d array)      y : the map to fit
                         y.shape should be (len(x[0]),len(x[1]))
```

8.18.9 fitBase2DGauss(mapArray,x,y,err=1.0,fwhm=11.0,gradient=1):

fitBase2DGauss < Timing: < Utilities.py

```
NAM: fitBase2DGauss (method)
DES: fits a 2D Gaussian + 1st order base surface
INP: (arrays) (x,y,mapArray,err)  : the data to fit (arrays of same dimension(s))
     (2els arrays) sizeX/Y : alternative to the x/y array, this limit the size of th
                             mapArray given as regular gridding between the center
                             of the two extreme pixels
     (float)         fwhm  : the first guess for the fwhm
     (logical)   gradient  : should we also fit a gradient in the map (default no) ?
OUT: a dictionnary containning the results of the fit
     check 'status' and 'errmsg' to see if the fit was done correctly
     then for each parameters (see the parname variable below) you have the
     'value' 'error' and 'limits' for the fit
```

8.18.10 cropped_circular_gaussian(p,position,threshold=3):

cropped_circular_gaussian < Timing: < Utilities.py

```
NAM: cropped_circular_gaussian
DES: compute a cropped circular gaussian with intensity=1
     defined by the parameter p wihtin the position an a given threshold given in n*
     position should be a list of 2 arrays of the same dimension defining the map
```

8.18.11 gaussian(r2,sig2):

gaussian < Timing: < Utilities.py

Attention: DES is missing! Please check the source code.
TXT:

```
INP: r2 = _array_ of distances^2, sig2 = sigma^2, related to Gaussian width
```

8.18.12 distsq(x1,y1,x2,y2):

distsq < Timing: < Utilities.py

```
NAM: distsq (function)
DES: returns distance squared between two points
INP: (float) x1,y1,x2,y2: coordinates of the two points
OUT: (float) distance^2
```

8.18.13 solvePoly(order,dataX,dataY):

solvePoly < Timing: < Utilities.py

```
NAM: solvePoly (function)
DES: perform polyomial interpolation: solve linear system
     dataY = P_n(dataX)
```

```
INP: (int) order : polynomial degree
     (flt arrays) dataX/Y : system to solve
OUT: (flt array) coeff : polynomial coefficients
```

8.18.14 module2(c):

module2 < Timing: < Utilities.py

```
NAM: module2 (function)
DES: return squared module of a complex number
INP: (complex) c: complex number
OUT: (float) |c|^2
```

8.18.15 mod_fft(z,optimize=0):

mod_fft < Timing: < Utilities.py

```
NAM: mod_fft (function)
DES: return squared module of the Fast Fourier Transform of an input sequence,
     limited to positive frequencies (assumes the input is real)
INP: (flt array) z: 1D array on which to compute FFT
     (logical) optimize: if true, optimize the computation time by limiting the
                         input array to its first p elements, where p is the
                         largest power of 2 less than the number of elemenets
OUT: (flt array) squared modules of FFT(z)
```

8.19 fUtilities (Fortran module)

8.19.1 ksmooth(image,ii,jj,kernel,ngg)

ksmooth < Module name missing < fUtilities.f90

```
! NAM: ksmooth (subroutine)
! DES: Smooth an image with a given kernel
!      the kernel should have a squared support with odd dimension
! INP:
!      image : input rank-2 array('f') with bounds (ii,jj); ii>1,jj>1
!      kernel  input rank-2 array('f') with bounds (ngg,ngg); ngg have to be odd and >1
! OUT:
!      image : rank-2 array('f') with bounds (ii,jj).
!
```

History:

```
! NAM: safeExp (subroutine)
! DES: Attempt to create a fast exponential for python
```

8.20 BoGLi modules and classes

List of Figures

1.1 The APEX telescope at Chajnantor, Images from the APEX handover, Nov. 2003 . . 1
5.1 Channel Map of source 00388+6312 with wedge 39
6.1 Example 1 of graphics produced using Plot.plot 50
6.2 Example 2 of graphics produced using Plot.plot 51
6.3 Example 3 of graphics produced using Plot.plot 51
6.4 Example of graphics produced using MultiPlot.plot 53
6.5 Example 1 of graphics produced using Plot.draw 54
6.6 Example 2 of graphics produced using Plot.draw: drawing contours 54
6.7 Example of graphics produced using MultiPlot.draw 56
7.1 Plotting the Signal for channels in the range. 69
7.2 Plotting the Mean values of signal. 70
7.3 Plotting the RMS values of signal. 71
8.1 Pointing 121
8.2 Pointing 122

List of Tables

1 3
2 3
3 4
4 List of commands with abbreviations 44
5 List of example BoGLi commands. 46
6 List of all float and integer parameters. 62